UNIVERSIDAD COMPLUTENSE DE MADRID
FACULTAD DE INFORMÁTICA
Departamento de Ingeniería del Software e Inteligencia Artificial

TESIS DOCTORAL

Técnicas para la optimización del análisis automático de vulnerabilidades en aplicaciones web

MEMORIA PARA OPTAR AL GRADO DE DOCTOR

PRESENTADA POR

Fernando Román Muñoz

Director

Luis Javier García Villalba

Madrid, 2018

Técnicas para la Optimización del Análisis Automático de Vulnerabilidades en Aplicaciones Web

TESIS DOCTORAL

Memoria presentada para obtener el título de
Doctor por la Universidad Complutense de Madrid
en el Programa de Doctorado en Ingeniería Informática

Fernando Román Muñoz

Director

Luis Javier García Villalba

Facultad de Informática
Universidad Complutense de Madrid

Madrid, Noviembre de 2017

Tesis Doctoral presentada por el doctorando Fernando Román Muñoz en la Facultad de Informática de la Universidad Complutense de Madrid para la obtención del título de Doctor por la Universidad Complutense de Madrid en el Programa de Doctorado en Ingeniería Informática.

Terminada en Madrid el 5 de Noviembre de 2017.

Título:

Técnicas para la Optimización del Análisis Automático de Vulnerabilidades en Aplicaciones Web

Doctorando:

Fernando Román Muñoz (froman@pdi.ucm.es)
Departamento de Ingeniería del Software e Inteligencia Artificial
Facultad de Informática
Universidad Complutense de Madrid
28040 Madrid, España

Director:

Luis Javier García Villalba (javiergv@fdi.ucm.es)

Esta tesis doctoral ha sido realizada dentro del grupo de investigación GASS (Grupo de Análisis, Seguridad y Sistemas, grupo 910623 del catálogo de grupos reconocidos por la UCM) como parte de las actividades del proyecto de investigación RAMSES (Internet Forensic Platform for Tracking the Money Flow of Financially-Motivated Malware) financiado por la Comisión Europea dentro del Programa Marco de Investigación e Innovación Horizonte 2020 (H2020-FCT-2015/700326-RAMSES).

A mi esposa Rosa, a mi hijo Miguel y a mi hija Teresa.

Agradecimientos

En primer lugar, estaré siempre agradecido a Javier por haberme dado la posibilidad de realizar esta tesis. Sin su ayuda, confianza y motivación, además de su capacidad de organización, esta tesis no habría sido posible. También le agradezco todos los medios que ha puesto a mi disposición para facilitar la comunicación a distancia y la realización de las tareas necesarias para el desarrollo de este trabajo. Quiero agradecer también la ayuda que me ha prestado Ana Lucila de manera desinteresada a lo largo de todo el proceso de elaboración de esta tesis.

Agradezco la colaboración de todos los miembros del Grupo de Análisis, Seguridad y Sistemas (GASS), en especial a Esteban, porque su apoyo en todos los ámbitos de la investigación ha supuesto para mí una gran ayuda para la realización de esta tesis.

No puede faltar el agradecimiento a mi familia por su apoyo y paciencia: a mis padres, Isidoro y Áurea, a mi hermano Eduardo, a mi esposa Rosa, a mi hijo Miguel y a mi hija Teresa.

Esta tesis doctoral ha sido realizada dentro del grupo de investigación GASS (Grupo de Análisis, Seguridad y Sistemas, grupo 910623 del catálogo de grupos reconocidos por la UCM) como parte de las actividades del proyecto de investigación RAMSES (Internet Forensic Platform for Tracking the Money Flow of Financially-Motivated Malware) financiado por la Comisión Europea dentro del Programa Marco de Investigación e Innovación Horizonte 2020 (H2020-FCT-2015/700326-RAMSES).

Índice General

Índice de Figuras

Índice de Tablas

Índice de Algoritmos

Lista de Acrónimos

AJAX	*Asynchronous Javascript And XML*
API	*Application Programming Interface*
BSQLI	*Blind SQL Injection*
CAPEC	*Common Attack Pattern Enumeration and Classification*
CAPTCHA	*Completely Automated Public Turing test to tell Computers and Humans Apart*
CMDExec	*Executing Command Prompt*
CMS	*Content Management System*
CSRF	*Cross-Site Request Forgery*
CSV	*Comma-Separated Values*
CVE	*Common Vulnerabilities and Exposures*
CWE	*Common Weakness Enumeration*
CWE25	*CWE/SANS Top 25 Most Dangerous Software Errors*
DOM	*Document Object Model*
DoS	*Denial of Service*
DVWA	*Damn Vulnerable Web App*
FI	*File Inclusion*
HTML	*Hypertext Markup Language*

HTTP	*Hypertext Transfer Protocol*
HTTPS	*Hypertext Transfer Protocol Secure*
IDS	*Intrusion Detection System*
IEC	*International Electrotechnical Commission*
ISO	*International Organization for Standardization*
JSON	*Javascript Object Notation*
LDAP	*Lightweight Directory Access Protocol*
LFI	*Local File Inclusion*
MVC	*Modelo–Vista–Controlador*
NIST	*National Institute of Standards and Technology*
NISTSP	*NIST Special Publication 500-269*
NTLM	*NT Lan Manager*
OWASP	*Open Web Application Security Project*
OWASPT10	*OWASP Top Ten*
OWASPTG	*OWASP Testing Guide*
OWASPTGv3	*OWASP Testing Guide v3*
OWASPTGv4	*OWASP Testing Guide v4*
PDF	*Portable Document Format*
PHP	*Hypertext Preprocessor*
PXSS	*Persitent Cross-Site Scripting*

REST	*Representational State Transfer*
RFI	*Remote File Inclusion*
RPC	*Remote Procedure Call*
RSQLI	*Reflected SQL Injection*
RTF	*Rich Text Format*
RXSS	*Reflected Cross-Site Scripting*
SAMATE	*Software Assurance Metrics And Tool Evaluation*
SCADA	*Supervisory Control And Data Acquisition*
SDK	*Software Development Kit*
SNORT	*Network Intrusion Detection and Prevention System*
SOAP	*Simple Object Access Protocol*
SQL	*Structured Query Language*
SQLI	*SQL Injection*
SRD	*Samate Reference Dataset*
SSI	*Server-Side Include*
SSL	*Secure Sockets Layer*
SSQLI	*Stored SQL Injection*
TF-IDF	*Term Frequency-Inverse Document Frequency*
URL	*Uniform Resource Locator*
WAF	*Web Application Firewall*
WASC	*Web Application Security Consortium*
WASCTC	*WASC Threat Classification*
WASSEC	*Web Application Security Scanner Evaluation Criteria*
WEBAPPSEC	*Threat Classification Taxonomy Cross Reference View*

WSDL	*Web Services Description Language*
WVS	*Web Vulnerability Scanner*
XML	*eXtensible Markup Language*
XSS	*Cross-Site Scripting*
ZAP	*Zed Attack Proxy*

Abstract

At present, the use of web applications has spread across many areas of daily life, with many potential users able to readily access it. It can therefore be exposed to malicious attacks. For example, one may try to bypass the predefined stream or enter unexpected values. These applications are developed with the aim of providing some functionality, which risks neglecting the security aspect. To mitigate the risks, it is common to perform a vulnerability analysis on web applications during the development life cycle.

Methods for detecting vulnerabilities in web applications can be of two types: static if they parse the source code of the application or dynamic if they analyze the application executing it. In static methods, the source code of the application is analyzed to detect fragments that correspond to known vulnerability patterns. In dynamic methods, the application is executed and from a real or simulated browser, values are sent to the application and the responses are recorded and analyzed to try to recognize vulnerabilities in them. The latter are the most frequent. Dynamic vulnerability analysis consists of two stages: the first is called passive or crawling phase and the second is called active phase. In the passive phase, it tries to locate the greatest possible number of entry points of the application. In the active phase, certain tests are performed on the entry points previously located to try to find vulnerabilities. The initial scanning phase is critical in the scanning process as only vulnerabilities can be searched for in content that has been detected. In other words, in the entry points to the application (pages, forms, fields, headers, etc.) that, even if they exist, were not previously detected in the crawl phase, no tests will be performed to see if it contains vulnerabilities. To perform the dynamic analysis an automatic tool is usually used that follows these two phases from the URL and eventually valid credentials.

Current tools do not fully perform the first phase of crawling primarily for two reasons: firstly, because these tools are not able to properly fill out forms, especially those with very strong constraints in their fields, and secondly because they often do not execute correctly Client code that the application sends to the browser, such as JavaScript code.

With respect to the second phase of testing and analysis, the main weakness detected in the use of these tools lies in the characteristics they must have and in the method to evaluate them, since there are no defined and accepted criteria about the functionalities Which must have the tools, i.e., which vulnerabilities to test and detect if this is the case. Also, there is no information about the tests that the tools do when analyzing an application. The traditional method of testing and comparing scanning tools is to attempt to detect the vulnerabilities in a vulnerable application and to analyze the resulting reports. The set of vulnerabilities to detect changes from one job to another, so you cannot contrast the results, and do not study what tests perform the tools.

In this work, solutions are provided for the deficiencies of the tools in the tracing phase and, for the test and analysis phase, the vulnerabilities that they should try are indicated. The first contribution of this thesis translates into two improvements in the tracking phase: on the one hand, a new method is created to find values of the fields of the forms that the application will accept, being able to advance in its defined operating flow and, On

the other hand, a mechanism is implemented by which an automatic tool can execute the client code in the same way as a browser used by a user.

The second contribution, related to the test and analysis phase, is the compilation of a complete classification of vulnerabilities, which groups the current classifications and can also be kept up to date. These vulnerabilities are what the tools should be able to detect.

The third contribution is the development of a vulnerable application that has implemented most web vulnerabilities and in which, by way of example, a toolkit is tested with it to determine its detection capabilities.

The fourth and final contribution is related to the previously mentioned fact that it is not known what the tools do. To try to determine the actions of the tools on the applications, a solution is implemented, based on placing an intermediate element between tool and application, which says what tests the tools do and is compared with the vulnerabilities of the application and with the final report of the application. tool. In this way, it is known if the tools prove everything they should and can and do report everything they have tried and detected.

Keywords: Analysis Tools, Detection Capability, Dynamic Vulnerability Analysis, Forms Values, Vulnerabilities, Vulnerability Classification, Vulnerability Detection, Vulnerability Scanner, Vulnerable Application, Web Application Crawling.

Resumen

En la actualidad el uso de las aplicaciones web se ha extendido a muchos ámbitos de la vida cotidiana, estando disponibles en cualquier momento a un gran número de usuarios potenciales y, por lo tanto, expuestas a ataques malintencionados o no. Por ejemplo, uno puede intentar saltarse el flujo predefinido o introducir valores no esperados. Estas aplicaciones se desarrollan con el objetivo de proporcionar cierta funcionalidad, dejando en muchos casos la seguridad en un segundo plano. Para mitigar los riesgos es habitual realizar un análisis de vulnerabilidades sobre las aplicaciones web durante el ciclo de vida de su desarrollo.

Los métodos para detectar las vulnerabilidades en las aplicaciones web pueden ser de dos tipos: estáticos si analizan el código fuente de la aplicación o dinámicos si analizan la aplicación ejecutándola. En los métodos estáticos se analiza el código fuente de la aplicación para detectar fragmentos que se correspondan con patrones de vulnerabilidades conocidas. En los métodos dinámicos se ejecuta la aplicación y desde un navegador, real o simulado, se envían valores a la aplicación y se registran y analizan las respuestas producidas para intentar reconocer en ellas vulnerabilidades. Estos últimos son los más frecuentes.

El análisis dinámico de vulnerabilidades consta de dos etapas: una primera llamada fase pasiva o de rastreo y una segunda llamada fase activa. En la fase pasiva se pretende localizar el mayor número posible de puntos de entrada de la aplicación. En la fase activa se realizan determinadas pruebas sobre los puntos de entrada localizados anteriormente para intentar encontrar vulnerabilidades. La fase inicial de rastreo es crítica en el proceso de análisis, ya que sólo se podrán buscar vulnerabilidades en el contenido que se haya detectado. En otras palabras, en los puntos de entrada a la aplicación (páginas, formularios, campos, cabeceras, etc.) que, aun existiendo, no se han detectado previamente en la fase de rastreo, no se realizará ninguna prueba para ver si contienen vulnerabilidades. Para realizar el análisis dinámico se suele emplear una herramienta automática que sigue estas dos fases a partir de la URL y eventualmente unas credenciales válidas.

Las herramientas actuales no realizan por completo la primera fase de rastreo fundamentalmente por dos motivos: primero porque estas herramientas no son capaces de rellenar adecuadamente los formularios, especialmente aquellos que tienen restricciones muy fuertes en sus campos, y segundo porque a menudo no ejecutan correctamente el código de cliente que envía la aplicación al navegador, como, por ejemplo, el código JavaScript.

Respecto a la segunda fase, de prueba y análisis, la principal carencia que se detecta en el uso de estas herramientas se encuentra en las características que deben tener y en el método para determinar si las tienen, ya que no existe un criterio definido y aceptado sobre las funcionalidades que deben tener las herramientas, es decir, qué vulnerabilidades deben probar y detectar si es el caso. Asimismo, tampoco hay información sobre las pruebas que realmente hacen las herramientas cuando analizan una aplicación. El método tradicional de probar y comparar las herramientas de análisis consiste en intentar detectar con ellas una serie de vulnerabilidades en una aplicación vulnerable y analizar los informes resultantes. El conjunto de vulnerabilidades a detectar cambia de un trabajo a otro, por lo que no

se pueden contrastar los resultados, y no se estudian qué pruebas realmente realizan las herramientas.

En este trabajo se aportan soluciones para las carencias de las herramientas en la fase de rastreo y, para la fase de prueba y análisis, se indican las vulnerabilidades que deberían probar.

La primera contribución de esta tesis se traduce en dos mejoras en la fase de rastreo: por un lado, se crea un nuevo método para encontrar valores de los campos de los formularios que la aplicación vaya a aceptar pudiendo avanzar en su flujo de funcionamiento definido y, por otro lado, se implementa un mecanismo mediante el cual una herramienta automática puede ejecutar el código cliente de la misma forma en la que lo haría un navegador empleado por un usuario.

La segunda contribución, relativa a la fase de prueba y análisis, consiste en la compilación de una completa clasificación de vulnerabilidades, que agrupa las clasificaciones actuales y que además puede mantenerse actualizada. Estas vulnerabilidades son las que las herramientas deberían ser capaces de detectar.

La tercera contribución es el desarrollo de una aplicación vulnerable que tiene implementadas muchas de las vulnerabilidades web y, en la que, a modo de ejemplo, se prueba un conjunto de herramientas con ella para determinar sus capacidades de detección.

La cuarta y última contribución está relacionada con el hecho comentado anteriormente de que no se sabe qué hacen realmente las herramientas. Para intentar determinar las acciones de las herramientas sobre las aplicaciones se implementa una solución, basada en situar un elemento intermedio entre herramienta y aplicación, que dice qué pruebas hacen las herramientas y se compara con las vulnerabilidades de la aplicación y con el informe final de la herramienta. De esta forma se sabe si las herramientas prueban todo lo que deben y pueden y si informan de todo lo que han probado y detectado.

Palabras clave: Análisis Dinámico de Vulnerabilidades, Aplicación Vulnerable, Capacidad de Detección, Clasificación de Vulnerabilidades, Detección de Vulnerabilidades, Escáner de Vulnerabilidades, Herramientas de Análisis, Rastreo de Aplicaciones Web, Valores de Formularios, Vulnerabilidades.

Parte I

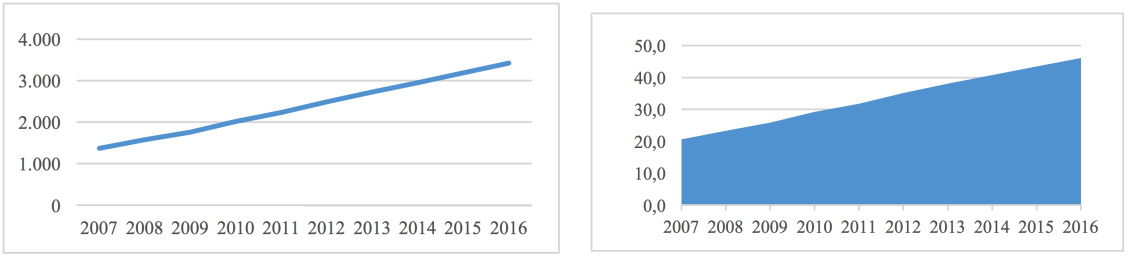
Descripción de la Investigación

Capítulo 1

Introducción

Actualmente, el uso de aplicaciones web en cualquier ámbito de la vida cotidiana aumenta año tras año. La proliferación de los dispositivos con capacidad de acceso a la red por los usuarios (ordenadores personales, teléfonos inteligentes, tabletas, etc.) o por sí mismos (electrodomésticos, domótica, control de la salud, etc.) ha provocado que cada vez más actividades habituales se trasladen a aplicaciones web. Concretamente, el número de usuarios de Internet creció un 7,5 % entre julio de 2015 y junio de 2016, situando el porcentaje de población mundial con acceso en un 46,1 % [Int17]. Para el caso de las aplicaciones web el número de sitios web desde enero de 2016 al mismo mes de 2017 ha aumentado en un 98 % [Net17].

En las Figuras 1.1(a) y Figura 1.1(b)) se muestra la evolución del crecimiento de usuarios con acceso a Internet a lo largo de los últimos 10 años.



(a) Millones de usuarios

(b) Porcentaje mundial

Figura 1.1: Personas con acceso a Internet

Una vez descrita esta visión general en cifras, de la magnitud de los usuarios de Internet y de las aplicaciones web, debe tenerse en cuenta que estas aplicaciones tienen una serie de características intrínsecas, con respecto a otro tipo de aplicaciones, que hace que su uso implique un mayor riesgo de ataque y posible éxito de éstos.

Estas características son:

- Disponibles públicamente.
- Uso de puertos permitidos en los elementos del cortafuegos, como el puerto 80 y el 443.
- Utilización del protocolo HTTP, que permite recibir datos del usuario de muchas formas: en la *Uniform Resource Locator* (URL), en el cuerpo de la petición, en cabeceras, a través de cookies, etc.
- Disponibilidad de técnicas y herramientas de ataque para explotar las vulnerabilidades en los entornos web.
- Ataques anónimos desde Internet sobre las aplicaciones web disponibles públicamente.
- Desarrollo ad-hoc y propietario de la mayoría de las aplicaciones web.
- Capacidades de autenticación y mantenimiento de sesiones en HTTP muy limitadas.
- Existencia de otros elementos potencialmente objetivo de los atacantes como cortafuegos, balanceadores, enrutadores, etc., además de la propia aplicación web.

De acuerdo al informe anual [Acu16] presentado por el fabricante de herramientas de análisis de aplicaciones web Acunetix, el 55 % de los sitios web en Internet presentan vulnerabilidades de severidad grave, habiendo sufrido un aumento del 9 % de abril de 2015 a marzo de 2016. Además, el número de sitios web con vulnerabilidades de severidad media se sitúa en el 84 %. Adicionalmente, el 16 % de los activos de la red perimetral también presentan al menos una vulnerabilidad de severidad media. El incremento del número de vulnerabilidades de un año a otro se debe principalmente a que no se tiene en cuenta la seguridad en las fases iniciales del desarrollo o incluso, habiéndola tenido, no se examinan lo suficiente. Además, para complicar más las cosas, no hay dos aplicaciones web iguales. Cada aplicación web incluye código desarrollado a medida, módulos, configuraciones, u otras personalizaciones, que sólo existen en esa aplicación.

En el informe sobre la seguridad en las aplicaciones web referente a 2016 [Sec16] de la empresa de seguridad WhiteHat Security, se indica que cada una de las aplicaciones web que ellos han analizado tienen entre 5 y 32 vulnerabilidades. También informan sobre el porcentaje de aplicaciones web que siempre son vulnerables, es decir, que están continuamente expuestas a algún tipo de ataque con éxito, según la industria a la que se refiera. En la Figura 1.2 se indica el porcentaje de aplicaciones siempre vulnerables de las industrias tenidas en cuenta.

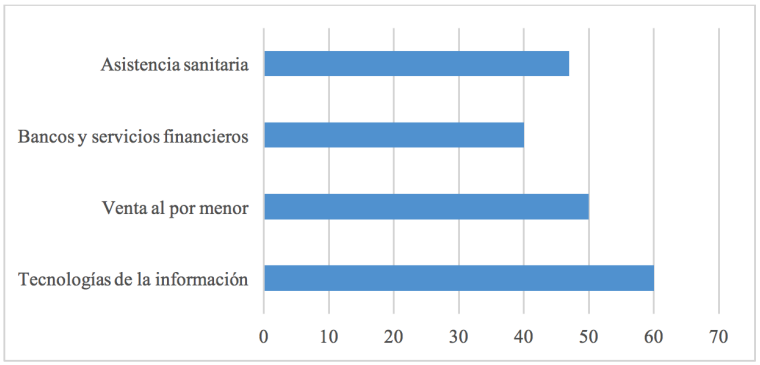


Figura 1.2: Aplicaciones web siempre vulnerables por industria

Para intentar corregir este comportamiento se ha afrontado el problema desde dos frentes: durante el proceso de diseño y desarrollo de la aplicación, estableciendo requisitos y verificaciones de seguridad que debe cumplir toda aplicación web y, posteriormente, finalizado el desarrollo antes o tras la puesta en producción de la aplicación web, realizando el análisis de la aplicación mediante auditorías de seguridad, análisis de vulnerabilidades o pruebas de intrusión.

El análisis de las aplicaciones web puede ser de caja blanca o de caja negra. El análisis de caja blanca o análisis estático, consiste en examinar el código fuente y la estructura de la aplicación [ND12] [SM15] buscando fallos que puedan implicar brechas de seguridad cuando vaya a ejecutarse, convirtiéndola en una aplicación vulnerable. El análisis del código se puede hacer mediante el uso de herramientas específicas o también de forma manual [MK15]. En el análisis de caja negra o dinámico, se analiza la seguridad de la aplicación de forma funcional [BBGM10] [ND12] [SM15] sin analizar el código fuente. Mediante el uso de herramientas automáticas o manualmente se buscan las posibles vulnerabilidades en la aplicación en ejecución.

Tanto el análisis estático como el dinámico se puede realizar de forma manual, de forma automática o, como es más habitual, de forma combinada o semiautomática. Para realizar un análisis dinámico de forma automática se usan una o varias herramientas que a partir de la URL de la aplicación y eventualmente un conjunto de credenciales válidas (usuario y contraseña, certificado digital de cliente, cookie de sesión, etc.) busca las vulnerabilidades de la aplicación.

El uso de herramientas de análisis dinámico reduce tiempo y esfuerzo dentro del ciclo de desarrollo de una aplicación y además permite enfocar mayores esfuerzos en tareas de seguridad más complejas [Bar11]. La configuración de este tipo de herramientas no es trivial, sobre todo para quien no está familiarizado con ellas [MK15] y aunque se usan habitualmente y proporcionan información valiosa, adolecen de varias carencias que es preciso solventar [GS11] [FK11].

1.1. Motivación

Actualmente se utilizan herramientas de análisis automático para analizar las aplicaciones web y determinar sus vulnerabilidades. A pesar de que en el mercado se pueden hallar distintas herramientas de análisis automático de vulnerabilidades en aplicaciones web, ninguna de estas herramientas es cien por cien efectiva. Esto implica que en la mayoría de las ocasiones es necesario realizar un análisis posterior manual para comprobar si los resultados de las herramientas son ciertos y para buscar, y encontrar en muchos casos, vulnerabilidades que las herramientas no han encontrado.

Por un lado, hay bastantes trabajos existentes en los que se evalúan y comparan estas herramientas automáticas para determinar sus capacidades de detección y los puntos en los que deberían mejorar. Por otro lado, no hay muchos trabajos previos en los que se indiquen que deberían detectar las herramientas ni qué pruebas hacen realmente sobre las aplicaciones web.

En este trabajo se da solución a los puntos de mejora de las herramientas y se profundiza en las características que deberían incluir las herramientas, proponiendo nuevos puntos de mejora. De esta forma se podrán reducir los recursos en tiempo y personas dedicados a las pruebas manuales, automatizando el análisis lo máximo posible.

1.2. Objetivos

El objetivo general de esta Tesis es solucionar los principales problemas y carencias de las herramientas actuales de análisis automático de vulnerabilidades en las aplicaciones web. Por ello, los objetivos específicos son:

- Implementar un método para encontrar valores de los campos de los formularios que las aplicaciones web vayan a aceptar como válidos.
- Implementar un mecanismo mediante el cual una herramienta automática pueda ejecutar el código cliente como lo haría el navegador.
- Obtener un método para determinar las vulnerabilidades que deberían detectar las herramientas unificando las clasificaciones actuales de vulnerabilidades web.
- Mantener actualizadass las relaciones entre las diferentes clasificaciones de vulnerabilidades web actuales.
- Elaborar una aplicación web vulnerable que contenga el mayor número de vulnerabilidades actuales.
- Determinar las pruebas que realmente hacen las herramientas con el fin de evaluar sus capacidades de detección y proponer mejoras que deberían implementar.

1.3. Identificación del Problema

Las aplicaciones web son un tipo de aplicaciones en donde intervienen principalmente, y entre otros componentes, un servidor web y un navegador web. El navegador envía peticiones al servidor y el servidor las responde. La mayoría del tráfico que se intercambia está en código HTML, que el navegador del cliente procesa para mostrárselo a los usuarios.

Como cualquier otro tipo de aplicación las aplicaciones web pueden contener vulnerabilidades, que podrían ser explotadas de forma intencionada por atacantes o no intencionada por usuarios legítimos. Para mitigar estas situaciones se pueden aplicar dos soluciones: eliminar las vulnerabilidades durante su desarrollo aplicando técnicas de desarrollo seguro, o bien, analizar la aplicación una vez terminada en busca de las vulnerabilidades, para después aplicar las correcciones necesarias.

Aunque lo más recomendable es desarrollar la aplicación desde el inicio teniendo en cuenta las posibles vulnerabilidades que puedan darse, el método más habitual suele ser analizar únicamente la aplicación una vez esté en producción. En la mayoría de los casos este análisis que se hace es de tipo dinámico, utilizando alguna herramienta automática.

Estas herramientas, aunque son de gran utilidad porque liberan gran parte del trabajo al especialista de seguridad que realiza el análisis, también presentan ciertas debilidades y carencias y hacen que las aplicaciones web no se analicen por completo ni teniendo en cuenta todas las opciones posibles. Las aplicaciones no son analizadas en toda su extensión principalmente porque este tipo de herramientas no son capaces de detectar todo el contenido y puntos de entrada de la aplicación. Además, no tienen en cuenta todas las opciones debido a que no incorporan características para detectar la mayoría de las vulnerabilidades, o al menos, no de forma adecuada.

En este trabajo se proponen soluciones para las debilidades y carencias indicadas. Para que la herramienta sea capaz de detectar más contenido de la aplicación se propone un método para obtener valores de los campos de los formularios que la aplicación pueda aceptar y permita avanzar en su flujo de funcionamiento. Para la detección de vulnerabilidades se propone un método para obtener una clasificación de las vulnerabilidades que deberían detectar y otro para determinar qué hacen realmente las aplicaciones, con independencia de lo que reporten.

1.4. Resumen de las Contribuciones

Los resultados de la investigación realizada en esta Tesis comprenden diversas contribuciones que han sido publicadas en diferentes revistas/congresos de alto impacto. Estas contribuciones se enmarcan en el área del análisis de vulnerabilidades de las aplicaciones web. Los métodos utilizados para probar y comparar las herramientas automáticas de análisis [RMnGV13] se presentan en el Capítulo 4. Un método para el preproceso de los formularios de aplicaciones web y para la obtención de un conjunto de valores correctos que se puedan usar durante la fase de rastreo [RMnGV13] [RMnGV15c] se describe en el Capítulo 7. El algoritmo y su implantación para la unificación de las clasificaciones de vulnerabilidades web [RMnGV14] [RMnGV15a] [RMnGV15b]

[RMnGV16] se describe en el Capítulo 8. Las características de la aplicación vulnerable a propósito desarrollada y su prueba con un conjunto de herramientas de análisis [RMnSCGV16] [RMnSCGV17] se explica en el Capítulo 9. Por último, la solución para determinar las capacidades de detección de las herramientas automáticas de análisis de vulnerabilidades web [RMnAVGV16] [RMnSCGV16] se muestra en el Capítulo 10.

En la Figura 1.3 se muestra (en verde) las mejoras introducidas en este trabajo al análisis de vulnerabilidades web (en azul) con herramientas automáticas.

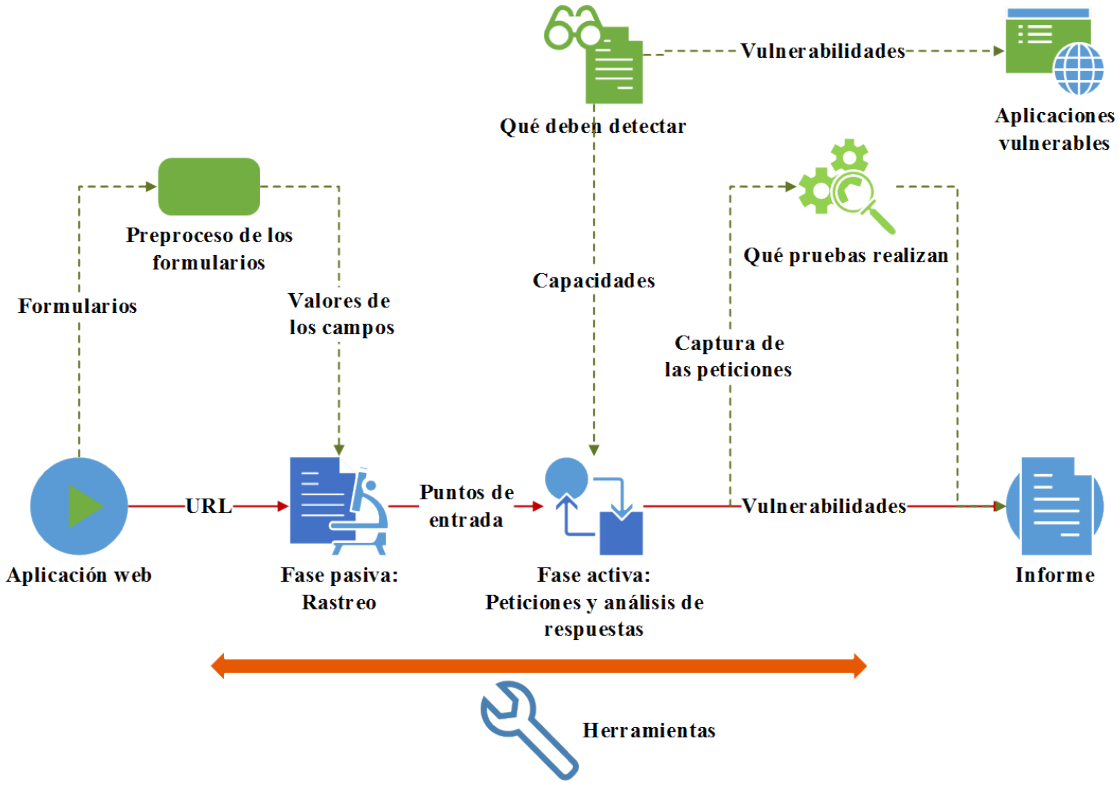


Figura 1.3: Esquema de las contribuciones

1.5. Estructura del Trabajo

Esta Tesis se estructura como sigue:

El Capítulo 2 describe los elementos y factores que intervienen en la seguridad de las aplicaciones web. Inicialmente, se presentan los conceptos de seguridad relacionados con las aplicaciones web. Posteriormente, se presta atención al desarrollo de las aplicaciones en general y, más específicamente, al de aplicaciones web, mostrando el modo de funcionar de estas últimas.

El Capítulo 3 presenta una visión general de las principales clasificaciones que existen actualmente de vulnerabilidades software, en especial las que se centran en aplicaciones web, describiendo también las relaciones que existen entre ellas y las vulnerabilidades

que contiene. A continuación, se presentan los tipos de aplicaciones web intencionalmente vulnerables que existen en la actualidad, indicando ejemplos en cada caso.

El Capítulo 4 introduce las razones que justifican la necesidad del análisis de vulnerabilidades en las aplicaciones web. Asimismo, se indican los principales tipos de análisis y sus características, exponiendo las fases del análisis dinámico [RMnGV13], que es el más utilizado. Finalmente, se hace una descripción panorámica de los distintos tipos de herramientas, describiendo sus debilidades y carencias. También se incluye, a modo de ejemplo, la descripción detallada de cuatro de estas herramientas, dos comerciales y dos de código libre.

El Capítulo 5 resume el estado del arte de las principales técnicas de evaluación de las capacidades de detección de las herramientas de análisis de aplicaciones web. Comienza con la exposición de las principales guías o estándares existentes para la evaluación de estas herramientas. A continuación, se muestran las distintas técnicas de evaluación, incluyendo una figura resumen de estas técnicas, para facilitar una visión general. Después se describen los principales trabajos relacionados, agrupados por técnica de evaluación empleada.

El Capítulo 6 presenta los problemas actuales de las herramientas automáticas de análisis de vulnerabilidades en aplicaciones web y las soluciones implementadas para solventar, o al menos mitigar, cada uno de ellos. Para solucionarnos se propone un algoritmo de preproceso de formularios web que ejecute el código de cliente, un método para obtener una clasificación actualizada de vulnerabilidades en aplicaciones web que son las que debería de detectar las herramientas automáticas, el desarrollo de una aplicación vulnerable a propósito y el uso de una herramienta auxiliar para determinar qué pruebas, de entre las que puede hacer y las que debe hacer cada herramienta, son las que realmente hacen.

El Capítulo 7 presenta la implementación y prueba del algoritmo definido para el preprocesado de formularios de aplicaciones web [RMnGV13] [RMnGV15c]. Con el preproceso se obtienen valores válidos para los campos de los formularios que permiten continuar en el flujo definido en la aplicación. Incluye la exposición del programa software desarrollado para ello, incluyendo la descripción de las tareas que realiza cada uno de los módulos que la componen. Se continúa indicando las pruebas realizadas, para finalizar con el resultado de las pruebas, que confirman la detección de más páginas que con las herramientas habituales.

El Capítulo 8 presenta el trabajo de unificación de las clasificaciones de vulnerabilidades web y de obtención de una clasificación que las contiene a todas ellas [RMnGV14] [RMnGV15a] [RMnGV15b] [RMnGV16]. Para obtenerla se utiliza la información que proporcionan los organismos que elaboran las herramientas e información disponible en Internet que relaciona unas clasificaciones con otras y también únicamente unas pocas vulnerabilidades con otras. Finalmente, se ha probado que esas relaciones obtenidas son ciertas.

El Capítulo 9 explica las características de la aplicación vulnerable a propósito seleccionada y mejorada [RMnSCGV17], describiendo las vulnerabilidades adicionales que se han implementado. El resultado de este capítulo también se muestra en [RMnSCGV16]. Seguidamente se describen las pruebas realizadas sobre esta nueva aplicación vulnerable

con un conjunto de herramientas automáticas. Para finalizar se indican los resultados de estas pruebas y las conclusiones que podemos extraer de ellas.

El Capítulo 10 presenta el resultado del análisis de las capacidades de las herramientas de análisis de vulnerabilidades en aplicaciones web [RMnAVGV16]. El análisis realizado permite, gracias al uso de un elemento intermedio entre herramienta y aplicación vulnerable, obtener detalles sobre qué hacen realmente las herramientas. De esta forma se llega a saber, de las vulnerabilidades que existen en las aplicaciones, cuáles prueban, pero no informan de ellas y cuáles no prueban, a pesar de contar con una funcionalidad para ello.

El Capítulo 11 expone las principales conclusiones de este trabajo, así como algunas posibles líneas futuras de investigación.

Capítulo 2

Seguridad en las Aplicaciones Web

El objetivo general de este capítulo es facilitar la comprensión de los elementos y factores que intervienen en la seguridad de las aplicaciones web. En primer lugar, se describen los conceptos de seguridad relacionados con las aplicaciones web. Posteriormente, se hace un repaso del ciclo de vida de desarrollo del software. Luego, se presta atención al desarrollo de las aplicaciones en general y, más específicamente, de aplicaciones web. Seguidamente, se muestra el modo de funcionar de estas últimas. El capítulo termina con una breve síntesis de lo expuesto en él.

2.1. Introducción

La seguridad de las aplicaciones web y, en general, de todas las aplicaciones software, debe tenerse en cuenta durante todo su ciclo de desarrollo. Cuanto antes y en mayor medida se impliquen los equipos de seguridad, más seguro será el producto resultante. Esta implicación pasa por conocer previamente las cuestiones de seguridad que pueden afectarles, y en el caso de las aplicaciones web, tener conocimiento de su funcionamiento, prestando especial atención al protocolo que define su funcionamiento.

A continuación, se muestran los conceptos relacionados con los riesgos que pueden afectar a las aplicaciones de software, las principales fases de su ciclo de desarrollo, indicando muy brevemente las tareas de seguridad que ha de realizarse en cada una de ellas y el funcionamiento de las aplicaciones web.

2.2. Conceptos de Seguridad

Cuando se habla de seguridad de la información habitualmente se tratan los ataques y la explotación de vulnerabilidades. También se suele hacer referencia al riesgo del sistema y de las medidas para mitigarlos. En otras ocasiones también se habla de amenaza, debilidad o control. Como puede verse existen diversos conceptos relacionados con la seguridad de los sistemas de información. Aunque cada uno de ellos es diferente, en muchas ocasiones no está clara su definición y se emplean indistintamente. Por lo tanto, es necesario explicar brevemente el significado de cada uno y las relaciones entre ellos.

Comenzando con las amenazas, éstas son cualquier evento que en caso de suceder tendría consecuencias negativas sobre el activo, en este caso el sistema de información o la aplicación de software. Las amenazas suelen explotar o hacer uso de debilidades, es decir, vulnerabilidades, para afectar negativamente al sistema. La existencia de vulnerabilidades implica cierto riesgo para la aplicación que se puede medir según el impacto negativo que le pueda causar a la aplicación y la probabilidad o frecuencia de que ocurra. Para proteger a los activos frente a las amenazas se pueden aplicar controles de seguridad que reducen o mitigan el riesgo. Adicionalmente tenemos los patrones de ataque que son descripciones de métodos para probar la explotación de vulnerabilidades y las características de las herramientas de análisis automático de vulnerabilidades que servirían para detectarlas.

En la Figura 2.1 se ilustran estas relaciones:

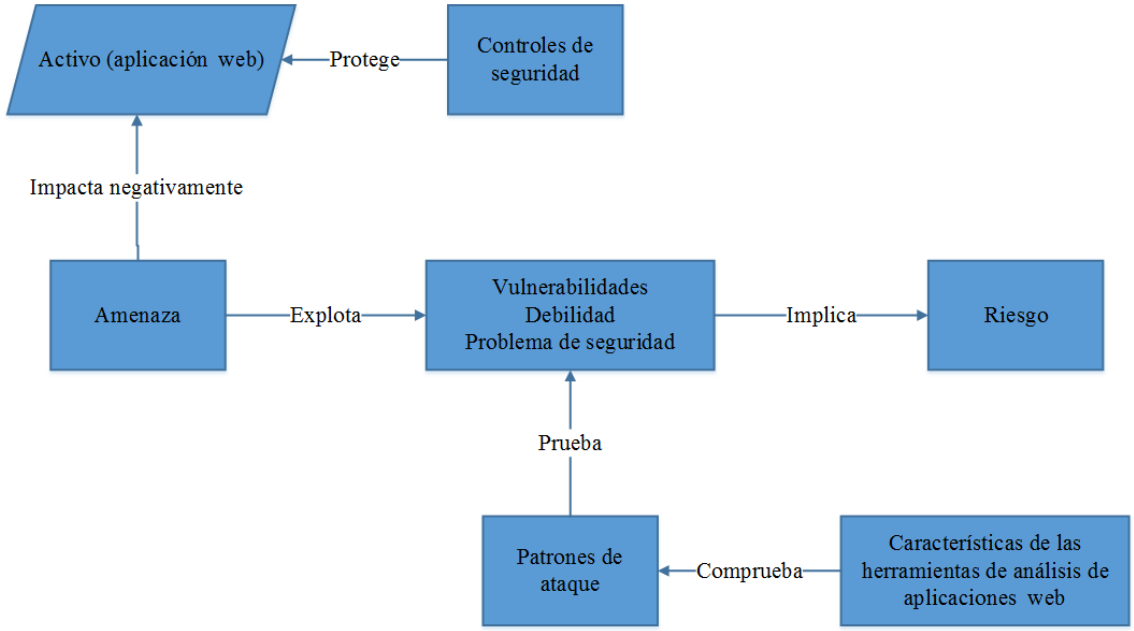


Figura 2.1: Conceptos de seguridad de la información

2.3. Ciclo de Vida del Desarrollo

El ciclo de vida de una aplicación software se compone de una serie de actividades relacionadas entre sí, que conducen a la elaboración de un producto de software. Existe una amplia variedad de procesos para el desarrollo de una aplicación o de un componente de software, pero todos deben incluir al menos las actividades que se muestran en la Figura 2.2. Todas las etapas del ciclo de vida del software se retro-alimentan entre sí. Esta información les permite identificar a tiempo posibles problemas y minimizar el coste en el mantenimiento del software. En la Figura 2.3 se presentan las principales recomendaciones de buenas prácticas de seguridad que se deberían de aplicar en cada una de las etapas del ciclo de vida de un software.

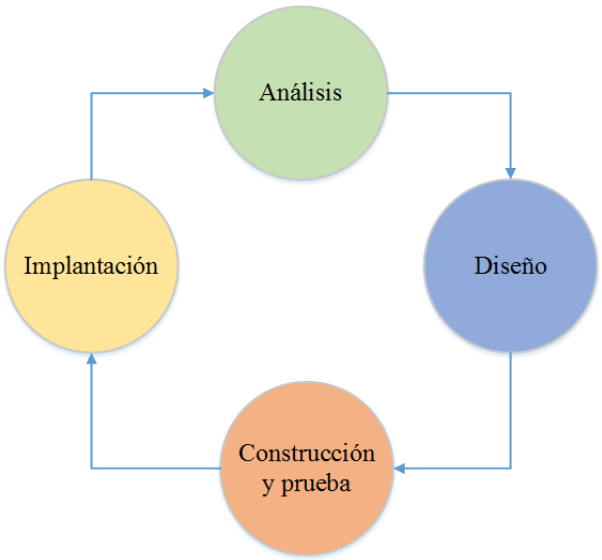


Figura 2.2: Actividades en el ciclo de vida de un software.

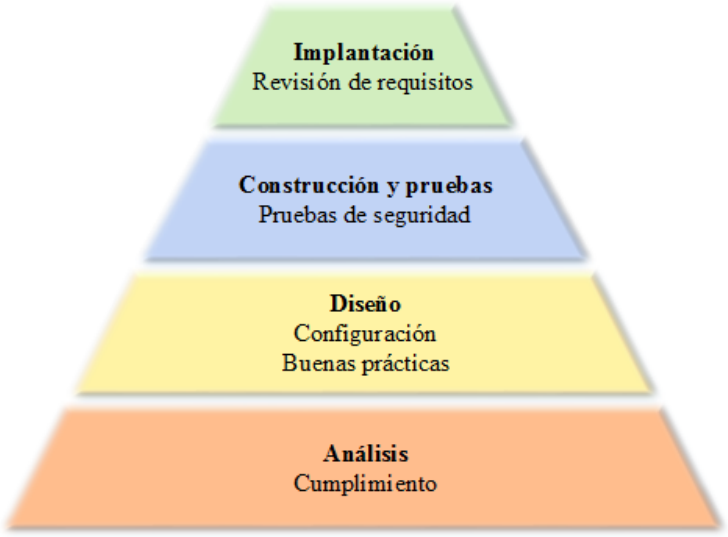


Figura 2.3: Seguridad durante el ciclo de vida de un software

A continuación se detallan las recomendaciones presentadas en la Figura 2.3.

2.3.1. Análisis del Sistema de Información

En esta etapa se describe a alto nivel el sistema de información, se delimita su alcance y se determinan sus requisitos funcionales y no funcionales. Dentro de estos últimos se encuentran los requisitos de seguridad. En esta fase además se especifican todas las interfaces entre el sistema y el usuario:

- Formatos de pantallas.
- Diálogos.
- Formatos de informes.
- Formularios de entrada.

Para determinar los requisitos de seguridad se debe tener en cuenta, si existe, la política de seguridad de la organización. A continuación, se deben incluir los requisitos derivados de cumplimientos legales o normativos que puedan afectar al sistema, según el tipo de información que vaya a tratar. Por ejemplo, leyes de protección de datos personales o de información confidencial, o la norma internacional *International Organization for Standardization* (ISO)/*International Electrotechnical Commission* (IEC) 27001 [Int05]. Se tienen en cuenta, si se han definido, los niveles de seguridad del sistema de información, por ejemplo:

- Autenticidad.
- Confidencialidad.
- Integridad.

A partir de estos requisitos se tiene, por ejemplo, el tipo de control de acceso a implementar, el mecanismo de cifrado de la información o los perfiles de usuarios a definir.

2.3.2. Diseño del Sistema de Información

En esta fase se obtiene el diseño detallado del sistema de información, incluyendo sus componentes y subsistemas, y se define la arquitectura tecnológica que va a dar soporte al sistema. A partir de dicha información se generan todas las especificaciones de construcción relativas al propio sistema, incluyendo las de seguridad, así como la especificación técnica del plan de pruebas.

Las especificaciones de seguridad se obtienen de analizar riesgos de seguridad, amenazas y vulnerabilidades, que puedan afectar a los componentes y a la arquitectura tecnológica establecida. Para ello se estudian los riesgos que plantea el despliegue del sistema de información a desarrollar según el diseño realizado y el entorno tecnológico previsto. Como resultado se determinan los posibles mecanismos de salvaguarda a implantar que minimicen los riesgos en función del impacto previsible de su materialización. Las salvaguardas pueden ser la aplicación de buenas prácticas de seguridad en el desarrollo para impedir ataques o recomendaciones de configuración en los elementos que dan soporte a la arquitectura como servidores, entornos de desarrollo, bases de datos, etc.

En esta fase de diseño se realiza también la especificación técnica del plan de pruebas. Este plan debe incluir el diseño específico de las pruebas de seguridad del sistema que se van a realizar y la manera de hacerse.

2.3.3. Construcción y Prueba del Sistema de Información

El objetivo de esta fase es la construcción y prueba de los distintos componentes del sistema de información. A partir del conjunto de especificaciones definidas en las fases anteriores se desarrollan también los procedimientos de operación y seguridad y se elaboran los manuales de usuario final y de explotación, si procede. Para conseguir dicho objetivo se prepara el entorno de desarrollo y se genera el código de cada uno de los componentes del sistema de información que se van realizando y a medida que se van terminando, se realizan las pruebas unitarias de cada uno de ellos y las de integración con los demás componentes, finalizando con las pruebas de implantación en el entorno de producción.

Dentro de las pruebas de implantación hay que incluir las de seguridad para comprobar la eficiencia de las salvaguardas implementadas, eliminando o mitigando los riesgos detectados en las fases anteriores. Se comprueba que el sistema cumple los requisitos legales y normativos, las recomendaciones de configuración y las buenas prácticas de seguridad en el desarrollo. También hay que comprobar que las salvaguardas no han introducido problemas funcionales adicionales.

Durante las pruebas de seguridad es habitual el uso de herramientas de análisis dinámico o pruebas de penetración, así como herramientas de análisis estático, que comprueban las posibles vulnerabilidades que pueda tener la aplicación desarrollada. Los incumplimientos o vulnerabilidades detectados se devuelven a las fases anteriores del ciclo para subsanarlos.

2.3.4. Implantación del Sistema

Esta fase tiene como objetivo la entrega y aceptación del sistema en su totalidad y llevar a cabo las actividades oportunas que permitan su ejecución en producción. Para ello se establece el plan de implantación y una vez realizado, se ejecutan las pruebas de implantación con la participación del usuario responsable.

Si el entorno donde se han realizado las pruebas de implantación del sistema no coincide con el entorno de producción, se debe comprobar de nuevo que se cubren las medidas de seguridad detectadas en las fases iniciales. Esto puede implicar realizar de nuevo las pruebas de seguridad.

Durante la ejecución en producción de la aplicación, ya con usuarios reales, puede ser necesario comprobar de nuevo la seguridad de la misma, bien por la introducción de nuevas vulnerabilidades o por la presencia de nuevos riesgos, amenazas o vulnerabilidades, no detectados inicialmente.

2.4. Aplicaciones Web

Un tipo particular de sistemas de información son las aplicaciones web. Estos sistemas se ejecutan en servidores web y son accedidos por los usuarios mediante navegadores a través de Internet o redes intranet. Estas aplicaciones son muy populares debido a lo cómodo que es el navegador web como cliente, a su independencia del sistema operativo y a la facilidad para actualizar y mantener la aplicación web sin necesidad de distribuir o instalar software a miles de usuarios potenciales.

2.4.1. Estructura de las Aplicaciones

Las aplicaciones suelen constar de una arquitectura multinivel y para facilitar su desarrollo es común el uso de patrones de diseño. Un patrón de diseño muy popular para el desarrollo de aplicaciones web es el *Modelo–Vista–Controlador* (MVC). Este patrón de diseño permite dividir una aplicación, o módulo de ella, en tres partes: Modelo, Vista y Controlador. La Figura 2.4 muestra cómo interactúan estos componentes lógicos:

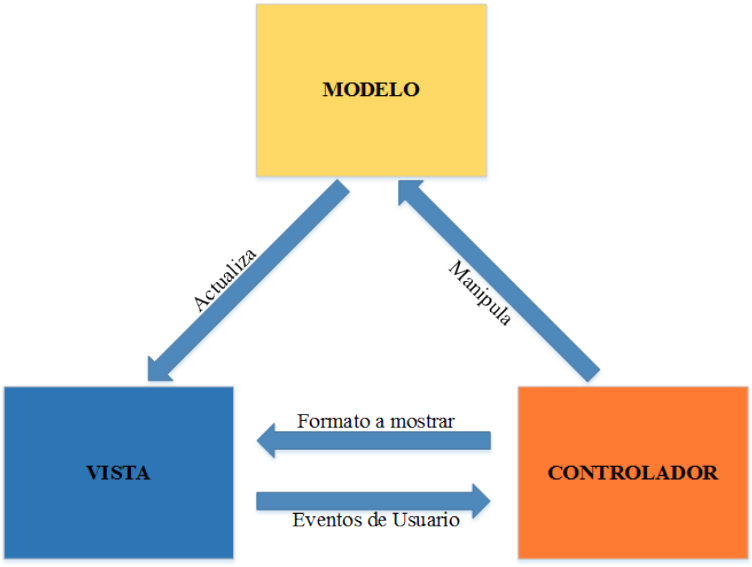


Figura 2.4: Patrón de diseño del modelo vista controlador

- **Modelo:** Gestiona los datos del sistema y las operaciones asociadas al mismo. Se puede considerar como el componente de persistencia en la aplicación.
- **Vista:** Define y gestiona como se muestran los datos de cara al usuario. Se puede considerar como la interfaz gráfica de la aplicación.
- **Controlador:** Gestiona la interacción del usuario con la aplicación, recibe los eventos generados por el usuario y responde a cada uno de ellos. Se puede considerar como la lógica de la aplicación.

En el desarrollo de algunas aplicaciones web se utiliza este patrón de diseño. Muchas infraestructuras digitales (*frameworks*) para el desarrollo de aplicaciones web como AngularJS o Backbone implementan un patrón MVC.

2.4.2. Estructura de las Aplicaciones Web

Aunque el modelo MVC se usa en el desarrollo de algunas aplicaciones web, en la mayoría de los casos estas últimas se desarrollan y subdividen en capas, siendo lo más habitual el dividirla en tres:

- **Capa de presentación:** es la capa con la que interactúa el usuario, presentando una interfaz gráfica. Captura la información que introduce el usuario, la filtra para evitar los errores de formato y la redirige a la capa de lógica de negocio. El filtrado que realiza debe incluir el filtrado de los datos para impedir el envío de información que pueda derivar en un ataque. Esta capa puede almacenar contenido estático como páginas HTML, imágenes o documentos.
- **Capa de negocio:** es la capa en la que se procesa la información enviada por el usuario y recibida de la capa de presentación que, tras ser procesada, remite las respuestas. En esta capa es donde se establecen las reglas que debe cumplir la información y los flujos de debe seguir. Se comunica con la capa de datos para recuperar, almacenar o modificar la información que gestione la aplicación.
- **Capa de datos:** es la capa donde se encuentran los datos y es la encargada de acceder a ellos. Está formada por uno o más gestores de bases de datos que reciben las solicitudes de recuperación, almacenamiento o modificación de la información, procedentes de la capa anterior.

La división en 3 capas (presentación, lógica de negocio y datos) no se corresponde exactamente con la que propone MVC. El Modelo de MVC se correspondería con la lógica de negocio y datos, la Vista sería únicamente el interfaz externo de la capa de presentación y el Controlador sería una mezcla de presentación y lógica de negocio.

2.4.3. Funcionamiento de las Aplicaciones Web

Partiendo del modelo de desarrollo en tres capas, esquemáticamente la forma de funcionar de una aplicación web es la que se describe a continuación y puede verse en la Figura 2.5.

- El usuario, a través del navegador, hace una petición HTTP al servidor web.
- El servidor web devuelve el contenido estático, y para el resto del contenido redirige la petición al servidor de aplicación. Si es necesario acceder a información se traslada la petición al servidor de la base de datos que realiza la operación necesaria.
- A continuación, el servidor de aplicación genera el contenido de la nueva página que es devuelta a través del servidor web al navegador del usuario. Esta página puede

incluir código ejecutable en el cliente, que también podrá generar dinámicamente parte de la página que se muestre al usuario.

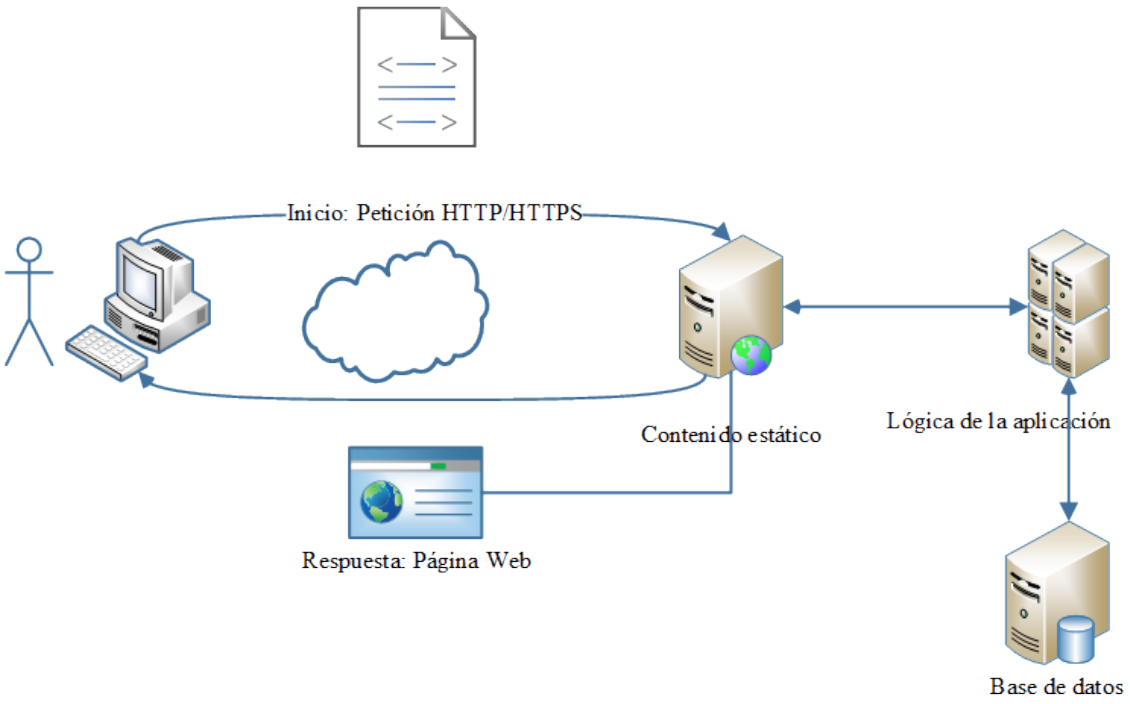


Figura 2.5: Diseño en tres capas

Como puede verse en la Figura 2.5, el usuario de la aplicación sólo se comunica con el servidor web ese es su único punto de acceso, por lo que no tiene información directa de lo que ocurre en los servidores de las dos capas siguientes. Su navegador envía y recibe peticiones HTTP. Las peticiones HTTP que genera el navegador ante la interacción del usuario tiene una estructura similar a la que se muestra en la Figura 2.6, en la que se ve un ejemplo de petición capturada con la herramienta BurpSuite.

- **Línea inicial**, que contiene principalmente el método de la petición (GET, POST, HEAD, etc.) y la URL solicitada.
- **Cabeceras** del mensaje, con información como la página desde la que se hace la petición, el tipo de navegador utilizado o el idioma.
- **Cuerpo** del mensaje, en el que va la información que el usuario envía al servidor. Este apartado es opcional, ya que, por ejemplo, si se usa el método GET los datos van en la URL, a diferencia de las peticiones usando el método POST, que sí tienen cuerpo del mensaje.

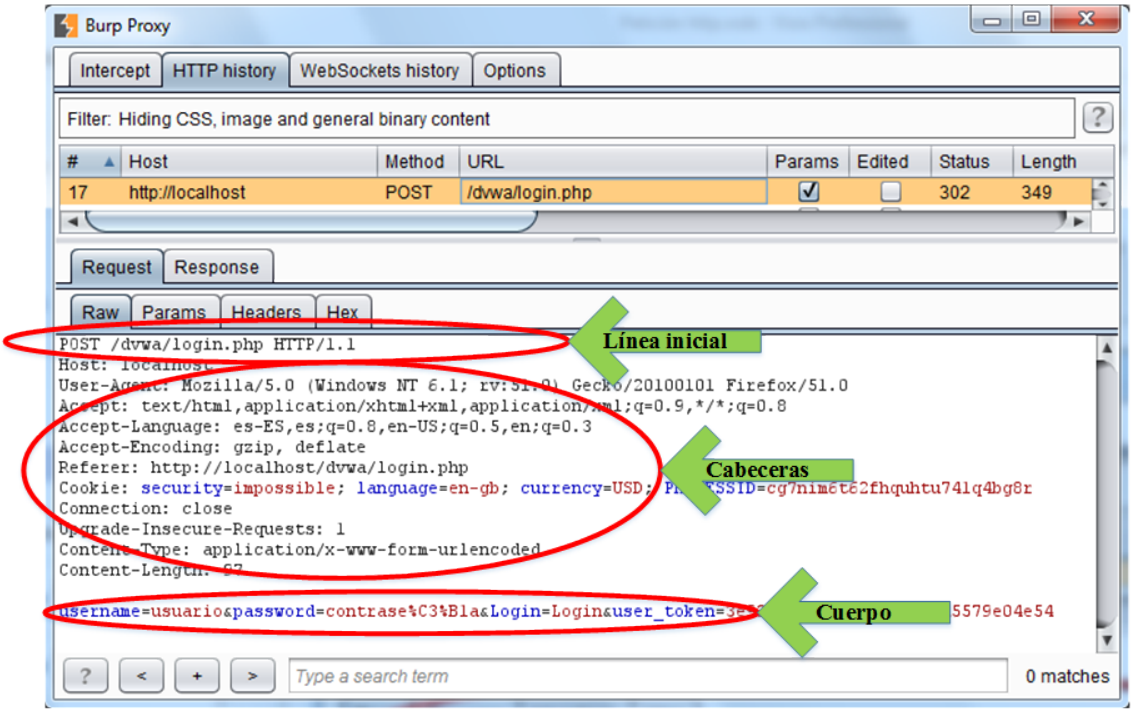


Figura 2.6: Ejemplo de petición HTTP

La estructura de las respuestas es similar a las de las peticiones. En la Figura 2.7 se muestra un ejemplo capturado con la herramienta BurpSuite:

- **Línea inicial**, que contiene la versión del HTTP usado seguido del código de respuesta del servidor.
- **Cabeceras** de la respuesta, que incluye información para el navegador, como el tamaño de la respuesta, la hora y la fecha o el tipo de servidor.
- **Cuerpo del mensaje**, que incluye los datos que el cliente ha solicitado, entendiendo por datos el contenido de lo que el usuario ha solicitado.

Es importante destacar que el protocolo HTTP es un protocolo sin estado, es decir, no guarda ninguna información sobre las conexiones anteriores. Pero como la mayoría de las aplicaciones web necesitan saber qué acción ha hecho anteriormente el usuario o en qué lugar de un flujo se encuentra, es necesario mantener el estado. Para ello se usan *cookies*, que es información que el servidor almacena en el navegador del usuario.

Una vez visto el funcionamiento básico de las aplicaciones web, es conveniente también indicar brevemente la forma de funcionar de algunas de sus principales características, como es el código JavaScript, incluyendo las peticiones JavaScript asíncrono, *eXtensible Markup Language* (XML), *Asynchronous Javascript And XML* (AJAX), los formularios web y el Modelo de Objetos del Documento (DOM).

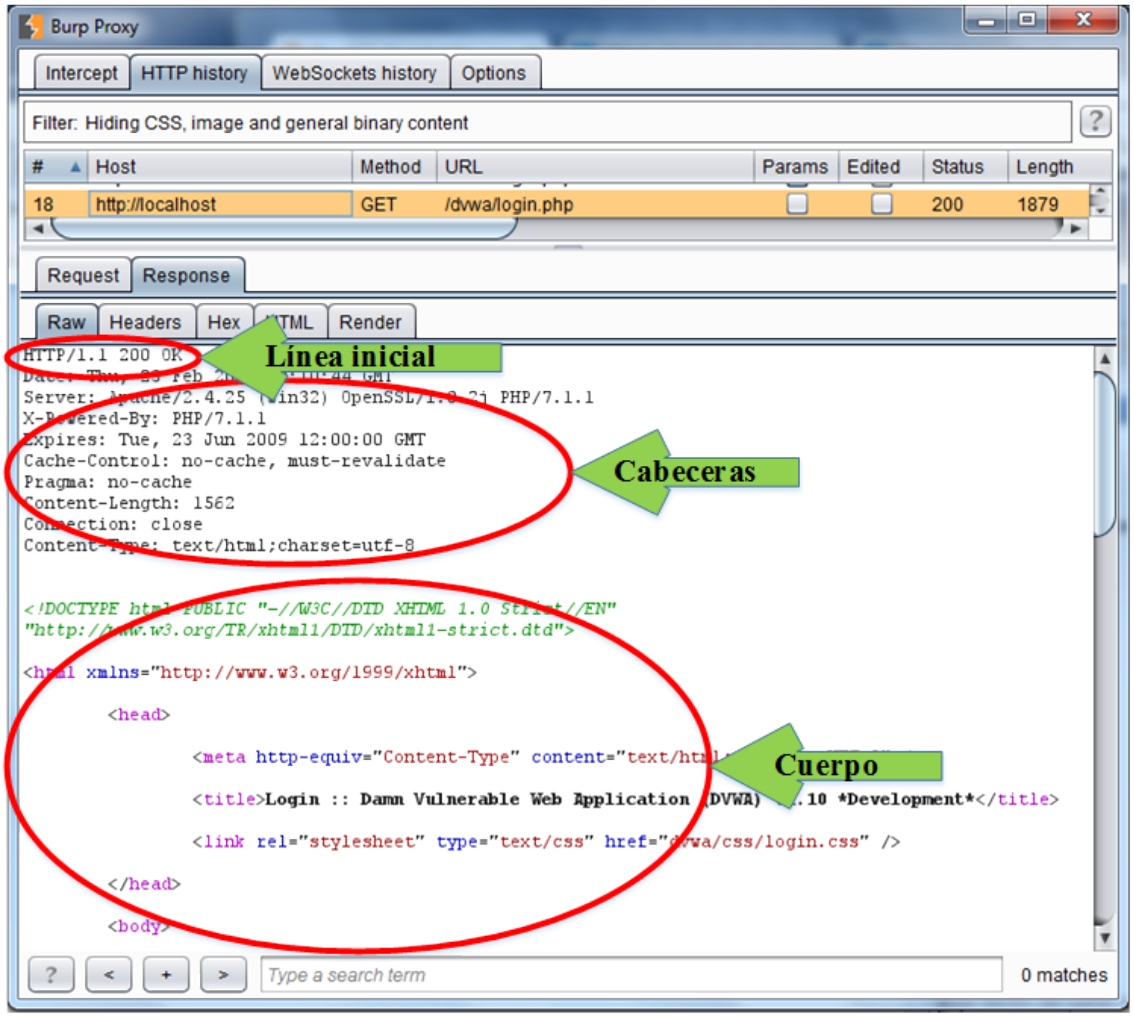


Figura 2.7: Ejemplo de respuesta HTTP

2.4.3.1. Código JavaScript

Como se ha indicado antes, la respuesta que devuelve el servidor al navegador del cliente puede que incluya código JavaScript. Este código puede ir incluido entre el propio código de la página, o referenciado y descargado por separado. El código JavaScript permite a las aplicaciones modificar la interfaz de usuario de acuerdo a sus características o preferencias, sin la intervención del servidor.

Además, el código JavaScript se emplea como parte de la técnica de desarrollo AJAX. El uso de AJAX consiste en que, al contrario que el uso principal indicado anteriormente, el código JavaScript sí genera una petición al servidor que devuelve su respuesta en formato XML. El objetivo de las peticiones AJAX es recuperar información del servidor, según las opciones seleccionadas por el usuario, eficientemente. En la Figura 2.8 se muestra el código JavaScript de un usuario que selecciona su país y la región y en lugar de cargar la página con todos los países y sus regiones, se envía una petición AJAX al servidor para recuperar solamente las regiones correspondientes al país seleccionado.

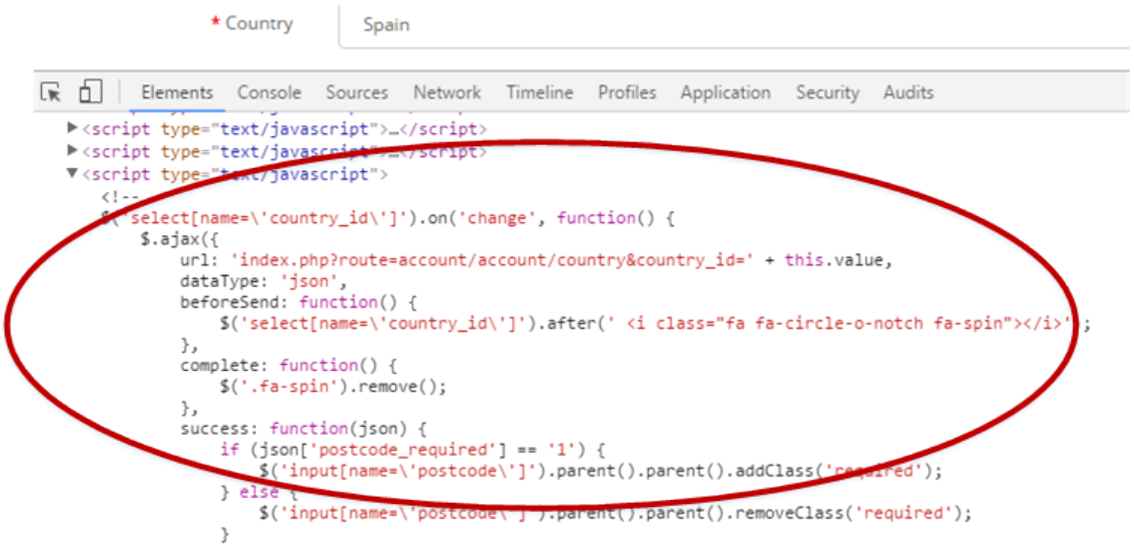


Figura 2.8: Petición JavaScript para recuperar regiones del país seleccionado

En la Figura 2.9 se puede observar la respuesta del servidor a la petición JavaScript anteriormente realizada.

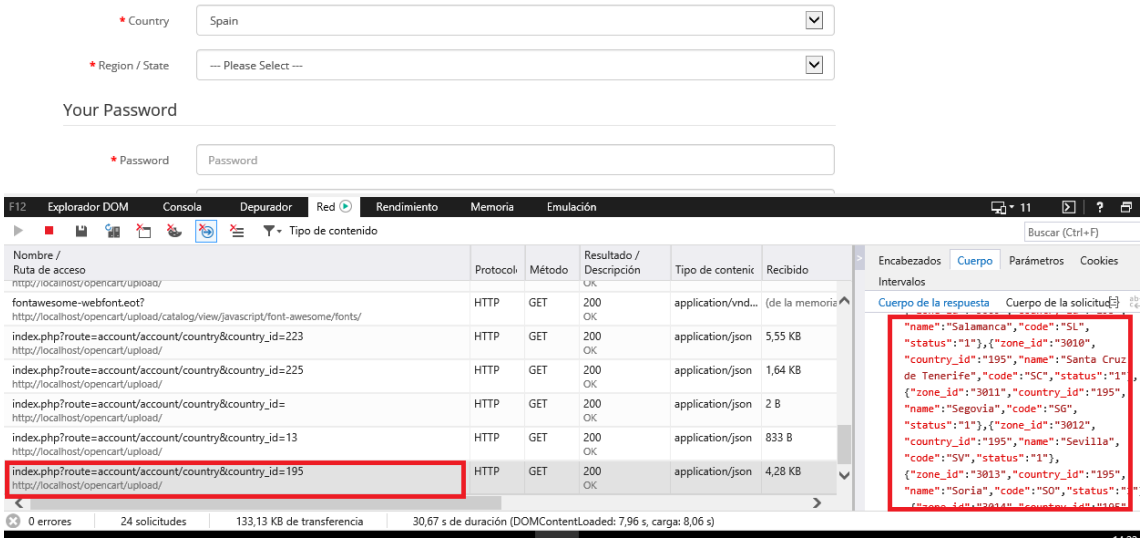


Figura 2.9: Respuesta XML con las regiones del país seleccionado

2.4.3.2. Formularios Web

Dentro de las páginas web es habitual que se encuentren formularios para que sus usuarios puedan enviar información al servidor para su procesamiento. Estos formularios están formados por dos zonas claramente diferenciadas, una con los campos y otra con los

botones de las acciones a realizar, siendo necesario que exista al menos un botón para el envío al servidor de la información introducida. Los campos pueden ser de varios tipos, pudiéndose clasificar principalmente en campos de selección, en los que el usuario elige una o varias de las opciones que se le presentan, y campos libres para que se introduzca texto o anexos.

Como se ve en la Figura 2.10, dentro del código HTML cada campo tiene, al menos, un identificador interno (en marrón) y un literal (en rojo), que es el que se muestra en el navegador. Aunque es recomendable que el identificador y el literal sean el mismo texto no tiene por qué ser así. En verde se muestra la acción a realizar cuando se pulsa el botón de "Login" que es el envío del usuario y la contraseña a la URL que se indica.

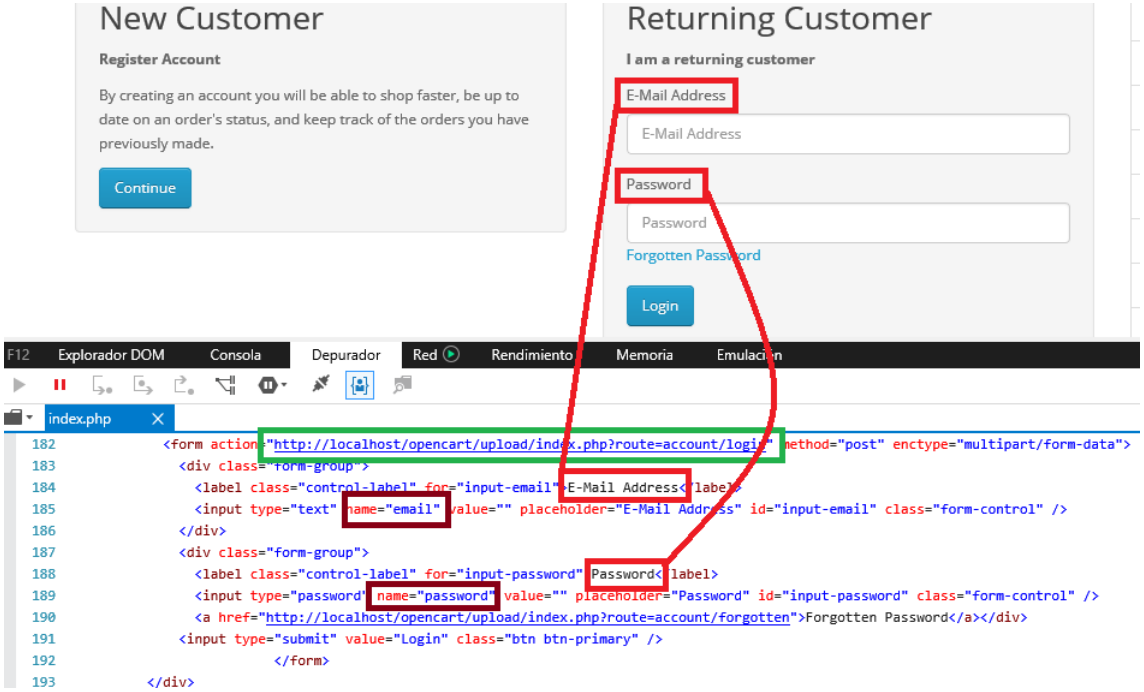


Figura 2.10: Código HTML de un formulario web

2.4.3.3. Modelo de Objetos del Documento

El Modelo de Objetos del Documento (DOM) es una interfaz de programación de aplicaciones para documentos HTML y XML. La parte más interesante del DOM en este trabajo es que crea la estructura jerárquica, en forma de etiquetas anidadas, de los elementos del código HTML que se esté manipulando.

Una representacion del DOM en el Algoritmo 1 sería el que se muestra en la Figura 2.11.

Algoritmo 1: Código HTML muy básico

```
1  <html>
2    <head>
3      <title>Titulo</title>
4    </head>
5    <body>
6      <h1>Contenido</h1>
7      <p>Mensaje</p>
8    </body>
9  </html>
```

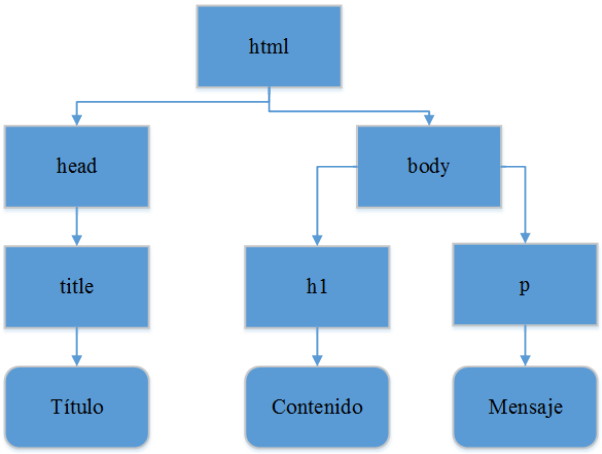


Figura 2.11: Representación de DOM

2.5. Síntesis del Capítulo

El objetivo de este capítulo ha sido resumir el funcionamiento de las aplicaciones web y su relación con la seguridad de la información.

Se ha comenzado con la exposición de los conceptos de seguridad y las relaciones entre cada uno de ellos. También se ha descrito el ciclo de vida del software, haciendo especial hincapié en las relaciones con la seguridad de la información presente en cada fase.

Finalmente, se ha realizado una descripción del funcionamiento de las aplicaciones web, con el objetivo de aclarar el funcionamiento del protocolo HTTP.

Capítulo 3

Vulnerabilidades en las Aplicaciones Web

En este capítulo se muestra una visión general de las vulnerabilidades existentes en aplicaciones web. En primer lugar, se hace una breve descripción de las principales clasificaciones de vulnerabilidades software, en especial las que se centran en aplicaciones web. Seguidamente se describen las relaciones que existen entre estas clasificaciones. Finalmente, se presenta brevemente los tipos de aplicaciones web intencionalmente vulnerables que existen indicando ejemplos en cada caso. El capítulo termina con una breve síntesis de lo expuesto en el mismo.

3.1. Generalidades

Varias organizaciones gubernamentales, comerciales o sin ánimo de lucro, han elaborado distintas clasificaciones para los diferentes tipos de vulnerabilidades que se pueden hallar en las aplicaciones informáticas, incluyendo las aplicaciones web. En estas clasificaciones se encuentran tanto las vulnerabilidades más populares y comunes como la *inyección Structured Query Language (SQL)* (*SQL Injection* (SQLI)), la secuencia de comandos en sitios cruzados (*Cross-Site Scripting* (XSS)), la falsificación de peticiones en sitios cruzados (*Cross-Site Request Forgery* (CSRF)), u otras no tan conocidas, como inyección *Lightweight Directory Access Protocol (LDAP)*, el uso de algoritmos de cifrado débiles o el escalado de privilegios.

No todas estas clasificaciones incluyen vulnerabilidades, algunas clasifican riesgos, otras amenazas y otras, por ejemplo, patrones de ataque. Pero, aunque es cierto que no clasifican según el mismo concepto, también es cierto que se pueden relacionar unas con otras. Por ejemplo, vulnerabilidades tan conocidas como el XSS y la SQLI suelen aparecer en todas ellas, con independencia del concepto con el cual se las clasifique.

3.2. Clasificaciones de Vulnerabilidades

Las clasificaciones de vulnerabilidades web más relevantes hasta el momento son las que realizan las organizaciones *Open Web Application Security Project (OWASP)*) y *Web Application Security Consortium (WASC)*). También el Instituto Nacional de Estándares y Tecnología (*National Institute of Standards and Technology* (NIST)) publica clasificaciones de vulnerabilidades web.

Además, existen organismos que mantienen listados de vulnerabilidades no sólo web. Entre ellos destacan el Instituto SANS y la Corporación MITRE. También hay otras clasificaciones realizadas por reconocidos expertos en seguridad de la información.

3.2.1. Fundación OWASP

La Fundación OWASP es una organización sin ánimo de lucro establecida en 2004. OWASP es una comunidad abierta dedicada a la concepción, desarrollo, adquisición, operación y mantenimiento de aplicaciones confiables para la seguridad de sistemas web y para el uso de cualquier organización que lo requiera. Toda la documentación y herramientas están disponibles y pueden ser utilizadas por cualquiera que lo necesite. Su propósito principal, de acuerdo a lo que declaran en su página web es "Ser la comunidad global más próspera que impulse la visión y la evolución de la seguridad y protección en el software mundial".

OWASP realiza periódicamente la clasificación de los fallos de seguridad más críticos para las aplicaciones web. Este documento es conocido como OWASP Top 10 (*OWASP Top Ten* (OWASPT10)) [The13] y consiste en una lista de los 10 riesgos de seguridad más críticos para las aplicaciones web de acuerdo al porcentaje de posibilidad de uso en un ataque y a la estimación del impacto en la aplicación atacada.

Los datos con los que realiza esta clasificación los obtiene de empresas consultoras y de fabricantes de herramientas especializadas en la detección de este tipo de vulnerabilidades. Actualmente se encuentra en desarrollo la versión 2016 y para este trabajo se ha considerado la información de la versión más actual de este listado, la versión 2013.

A continuación, se detallan cada una de las vulnerabilidades presentes en la clasificación OWASP10 de 2013.

1. **Inyección**: Esta vulnerabilidad puede ser de varios tipos, por ejemplo, de SQL, Sistema Operativo, LDAP o XML. Este tipo de vulnerabilidades permite al atacante enviar código malicioso a través de una aplicación hacia otro sistema, consiguiendo que el atacante ejecute comandos malintencionados en el sistema vulnerable o acceda a datos sin autorización.

2. **Fallo en autenticación y gestión de sesiones**: Es muy común que las funcionalidades relacionadas con la autenticación y gestión de sesiones de usuario en una aplicación no sean correctamente implementadas. Esta vulnerabilidad explota esos fallos y permite que un atacante pueda suplantar la identidad de cualquier usuario del sistema.

3. **XSS**: Esta vulnerabilidad ocurre cuando datos no confiables se envían a través de una aplicación web sin la adecuada validación y permiten que el atacante ejecute *scripts* maliciosos en el navegador de la víctima pudiendo incluso obtener sus credenciales de sesión, alterar la interfaz de la aplicación web o redirigir a la víctima hacia sitios maliciosos. Puede ser principalmente de dos tipos: reflejado (del inglés *Reflected Cross-Site Scripting* (RXSS)) si únicamente se modifica el código que se muestra a la víctima o persistente (del inglés *Persitent Cross-Site Scripting* (PXSS)) si se modifica la aplicación web.

4. **Referencia Directa de Objetos Insegura**: Esta vulnerabilidad está presente cuando el desarrollador de la aplicación expone en su código referencias hacia uno o varios objetos internos del sistema (fichero, directorio o claves de base de datos) de tal forma que el atacante pueda manipular estas referencias para acceder a datos sin autorización a través de ellas.

5. **Configuraciones de Seguridad Mal Implementadas**: La seguridad de una aplicación también depende de la correcta configuración de seguridad de cada uno de los componentes del entorno en donde se ejecute y de mantener todo el software actualizado para evitar vulnerabilidades derivadas de ello y que puedan ser utilizadas por parte de un atacante.

6. **Exposición de Datos Sensibles**: La correcta implementación de mecanismos seguros de gestión de los datos que administra una aplicación es fundamental y, más aún, si la aplicación trabaja con datos sensibles como números de tarjetas de crédito, documentos de identidad o credenciales de autenticación del sistema. Un atacante puede robar o modificar estos datos si es que no se encuentran adecuadamente protegidos. Los datos sensibles deben tener una protección adicional, como el cifrado.

7. **Ausencia de Control de Acceso a Funcionalidades**: Todas las aplicaciones web deberían implementar la verificación de nivel de acceso a sus funciones antes de mostrarlas en la interfaz del usuario y lo mismo del lado del servidor al cual cada funcionalidad accede. Si no existe este tipo de control, el sistema es vulnerable a que un atacante pueda acceder a funcionalidades para las cuales no está autorizado y ejecutarlas.

8. **CSRF**: Este tipo de vulnerabilidad obliga a que el navegador de la víctima envíe peticiones HTTP, en la cual están incluidas *cookies* de sesión y cualquier otra información de autenticación, hacia una aplicación web vulnerable y que ésta, a su vez, entienda cada petición como legítima de la víctima.

9. **Uso de Componentes Vulnerables**: Componentes como infraestructuras digitales, librerías y otro tipo de módulos de terceros suelen ejecutarse con alto nivel de privilegios. Si cualquiera de éstos contiene una vulnerabilidad que no ha sido corregida, puede ser utilizada por un atacante para acceder a datos sensibles, bloquear la aplicación o incluso el servidor en el cual se ejecuta. El uso de componentes con vulnerabilidades ya conocidas debilita la seguridad implementada

en la aplicación y aumenta la posibilidad de sufrir ataques por parte de un usuario malicioso.

10. **Redirección y Reenvío sin Validación**: Las aplicaciones web frecuentemente reenvían y redirigen sus usuarios hacia páginas y sitios web utilizando parámetros que determinan a dónde deben ir. Si la aplicación no cuenta con una adecuada validación de estos parámetros, un atacante puede utilizar esta vulnerabilidad de la aplicación web para redirigir al usuario a sitios maliciosos o acceder a datos para los cuales no esté autorizado.

OWASP también desarrolla la Guía de Pruebas de OWASP (*OWASP Testing Guide* (OWASPTG)), que describe un conjunto de pruebas que se pueden realizar sobre las aplicaciones web para detectar vulnerabilidades. La versión v3 (*OWASP Testing Guide v3* (OWASPTGv3)) de esta guía [OWA08] contiene 66 pruebas agrupadas en diez categorías:

- Recopilación de información
- Gestión de configuración
- Autenticación
- Gestión de sesiones
- Autorización
- Lógica de negocio
- Validación de datos
- Denegación de servicio
- Servicios web
- AJAX

La versión v3 es del 2008 e incluye la descripción de las vulnerabilidades, ejemplos y métodos de detección. No se indican métodos para mitigar su impacto, pero sí incluye referencia a otra información de interés de OWASP.

En 2014 esta guía se actualizó a la versión v4 (*OWASP Testing Guide v4* (OWASPTGv4)) [OWA14]. Al igual que la anterior versión, describe un conjunto de pruebas que se pueden realizar sobre las aplicaciones web para detectar vulnerabilidades. Esta versión tiene 88 pruebas distribuidas en 11 categorías.

Con respecto a la versión anterior se crean nuevas categorías y desaparecen otras como se observa en la Tabla 3.1.

Tabla 3.1: Categorías modificadas en OWASP v4

Categorías Nuevas	Categorías Eliminadas
Configuración y gestión del desarrollo	Gestión de configuración
Gestión de identidades	Gestión de sesiones
Gestión de errores	Denegación de servicio
Criptografía	Servicios web
Pruebas del lado del cliente	AJAX

3.2.2. Consorcio de Seguridad para Aplicaciones Web (WASC)

WASC es también un organismo sin ánimo de lucro, formado por un grupo internacional de expertos, profesionales y organizaciones de seguridad de la información, cuyo objetivo es desarrollar estándares de seguridad.

Esta organización desarrolla y mantiene una clasificación de 49 amenazas en aplicaciones web (*WASC Threat Classification* (WASCTC)) [Con10]. Se encuentra dividida entre debilidades y ataques, y también según el origen de la amenaza esté en el diseño, la implementación o en el desarrollo de la aplicación. Contiene descripciones y ejemplos. Su última versión es la 2.0 publicada en 2010.

En 2009 la organización WASC también elaboró *Web Application Security Scanner Evaluation Criteria* (WASSEC) [Con09], que incluye una lista de problemas de seguridad que una herramienta de análisis de aplicaciones web debe detectar. Incluye 55 problemas agrupados en varias categorías:

- Autenticación
- Autorización
- Ataque del lado del cliente
- Ejecución de comandos
- Revelación de información

3.2.3. Otras Clasificaciones

Además de los organismos OWASP y WASC, se pueden hallar más organizaciones e instituciones que realizan y mantienen clasificaciones de vulnerabilidades, únicamente web en algunos casos y de software en general en otros.

3.2.3.1. Instituto Nacional de Estándares y Tecnología (NIST)

Es una Agencia del Departamento de Comercio de los Estados Unidos. Como parte de su proyecto Herramientas de Evaluación y Métricas de Seguridad del Software (*Software Assurance Metrics And Tool Evaluation* (SAMATE)), surge en 2007 la guía *NIST Special Publication 500-269* (NISTSP) [BFOG08]. Esta guía describe las tareas que deben realizar

las herramientas de detección de vulnerabilidades web e incluye, además, una lista de 14 vulnerabilidades web que estas herramientas deben ser capaces de identificar. Las vulnerabilidades de esta lista se han elegido por su alta probabilidad de ser explotadas. Adicionalmente expone brevemente una metodología para mitigar estas vulnerabilidades, de estar presentes.

3.2.3.2. Corporación MITRE

MITRE es otra organización sin ánimo de lucro, financiada por el gobierno de Estados Unidos. Este organismo desarrolla y mantiene la clasificación Enumeración de Debilidades Habituales (*Common Weakness Enumeration* (CWE)) [Cor15b]. Esta clasificación es un conjunto de debilidades en todo tipo de software, no sólo en aplicaciones web. La versión existente en el momento de elaborar este documento es la 2.10, publicada en enero de 2017. Incluye 1005 debilidades, también se citan ejemplos, métodos de detección, de mitigación y referencias a otras clasificaciones de vulnerabilidades.

Conjuntamente con el Instituto SANS, MITRE ha desarrollado la clasificación *CWE/SANS Top 25 Most Dangerous Software Errors* (CWE25) [SAN11] de los errores software más peligrosos. Esta clasificación fue actualizada por última vez en 2011. Es un subconjunto de la clasificación CWE e, igualmente, incluye vulnerabilidades de todo tipo de aplicaciones. También incluye ejemplos y métodos de detección.

Otra clasificación que incluye vulnerabilidades presentes en todo tipo de software es el Listado Clasificador de Patrones Comunes de Ataque (*Common Attack Pattern Enumeration and Classification* (CAPEC)) [Cor15a], que, al igual que las dos anteriores, mantiene la corporación MITRE. Durante la elaboración de este documento la versión más actual es la 2.9, de enero de 2017, y contiene 503 patrones de ataque. Además, contiene también ejemplos y descripciones de flujos de ataque.

3.2.3.3. Otros

Además de las clasificaciones realizadas por los organismos indicados anteriormente, también existen expertos en seguridad de la información que mantienen y publican sus clasificaciones de vulnerabilidades. Entre ellos cabe destacar a Shay Chen, que compara herramientas de detección de vulnerabilidades web. Para ello ha elaborado una clasificación [Che12], con las 33 características de auditoría que las herramientas de análisis de aplicaciones web deberían tener. Se actualiza, por lo general, cada año e incluye referencias a la clasificación de WASC y a las guías de OWASP.

3.2.4. Comparación entre Clasificaciones

Las Tablas 3.2 y 3.3 resumen las características de estas listas de vulnerabilidades. En la Tabla 3.2 están las que clasifican sólo vulnerabilidades en aplicaciones web, y en la Tabla 3.3 las que incluyen vulnerabilidades en todo tipo de aplicaciones.

Tabla 3.2: Características de las clasificaciones de vulnerabilidades web

Clasificación	WASCTC	OWASPT10	OWASPTGv3	OWASPTGv3	NISTSP	WASSEC	SHAY
Desarrollado por	WASC	OWASP	OWASP	OWASP	NIST	WASC	Shay Chen
Concepto	Amenaza	Riesgo	Prueba de vulnerabilidad	Prueba de vulnerabilidad	Vulnerabilidad	Problema de seguridad	Características de auditoría
Número	49	10	66	88	14	55	33
Versión actual	2.0 (2010)	2013	2008	2014	1.0 (2007)	1.0 (2009)	2012
Descripción	Sí	Sí	Sí	Sí	Sí-brevemente	No	Sí-brevemente
Ejemplo	Sí	Sí	Sí	Sí	No	No	No
Detección	No	Sí	Sí	Sí	No	No	No
Mitigación	No	Sí	No	No	Sí-brevemente	No	No
Referencias	Sí	Sí	Sí	Sí	No	No	Sí

Tabla 3.3: Características de las clasificaciones generales de vulnerabilidades

Clasificación	CWE25	CWE	CAPEC
Proveedor	SANS-MITRE	MITRE	MITRE
Concepto	Debilidad	Debilidad	Patrón de ataque
Número	25	1003	463
Última versión	3.0 (2011)	2.10 (2017)	2.9 (2017)
Descripción	Sí	Sí	Sí
Ejemplos	Sí	Sí	Sí
Detección	Sí	Sí	Sí
Mitigación	No	Sí	Sí
Referencias	Sí	Sí	Sí

Como puede verse, las clasificaciones de vulnerabilidades no se actualizan muy frecuentemente; de hecho, únicamente dos clasificaciones de vulnerabilidades, CWE y CAPEC, se mantienen actualizadas más o menos de forma continua.

Como se ha mencionado anteriormente, aunque cada clasificación utiliza un concepto diferente para referirse a los problemas de seguridad, existen y se pueden establecer relaciones entre ellas. Por ejemplo, las dos vulnerabilidades más conocidas, XSS e SQLI, figuran como ataques en WASCTC (WASC-8 y WASC-19), como debilidades en CWE (CWE-79 y CWE-89), como pruebas en OWASPTGv4 (OTG-INPVAL-001 y OTG-INPVAL-005) o como riesgos en el OWASPT10 (A3 y A1). En cada caso se explica la información correspondiente; si es una prueba de seguridad se indica cómo realizarla; o si es un riesgo comó mitigarlo.

3.3. Relación entre las Clasificaciones de Vulnerabilidades Web

Para relacionar las vulnerabilidades de unas clasificaciones con las de otras, las propias organizaciones que desarrollan y mantienen las clasificaciones, o bien otros organismos o empresas dedicados a la seguridad de la información, realizan varios trabajos de asociación. Estas relaciones suelen incluir todas las vulnerabilidades de una clasificación relacionando aquellas que sean posibles con vulnerabilidades de otras clasificaciones. Las principales asociaciones entre clasificaciones son:

- **Denim Group** realizó en 2010 la vinculación entre WASCTC, CWE25 y OWASPT10 de 2004 y 2007.
- **Jeremiah Grossman** en 2009 relacionó la clasificación WASCTC con OWASPT10 de 2010.
- **Vista de referencia cruzada de la taxonomía de clasificación de amenazas (*Threat Classification Taxonomy Cross Reference View* (WEBAPPSEC))** [Con12] es una vista de la clasificación WASC, actualizada en 2013, que relaciona sus amenazas con CWE, CAPEC, OWASPT10 (de 2004, 2007 y 2010) y CWE25. Para ello usa la información de las relaciones indicadas anteriormente, Denim Group y Jeremiah Grossmans.
- **NISTSP** en su anexo A incluye una relación con CWE, OWASPT10 de 2007 y *Common Vulnerabilities and Exposures* (*Common Vulnerabilities and Exposures* (CVE)).
- **Securing Telligent Evolution** es un documento creado por Telligent en 2012, que describe las amenazas que se prueban en la plataforma Telligent Evolution, incluyendo la relación entre OWASPTGv3 y WASCTC.
- **SHAY** incluye también relaciones entre las vulnerabilidades de WASCTC y OWASPTGv3.

La información de todas estas relaciones se puede resumir para determinar qué clasificaciones están más relacionadas. La clasificación WASCTC está relacionada con otras 8 clasificaciones, CWE25 con 5 clasificaciones, OWASPT10 de 2007/2010 con 3 clasificaciones, y OWASPTGv3, CWE, CVE, y CAPEC sólo con 1 clasificación.

En la Tabla 3.4 se muestra un resumen de las características de las relaciones anteriormente indicadas. En la columna Clasificación se indica la clasificación que se ha usado como base para la vinculación.

Tabla 3.4: Relación entre clasificaciones de vulnerabilidades

Mapeo	Clasificaciones	Clasificación Maestra	Fuente Externa	Última Actualización
Denim Group	OWASPT10 (2004), OWASPT10 (2007), CWE25, WASCTC (en su versión anterior v1).	Todas	No	2010
WEBAPPSEC	WASCTC, CWE, CAPEC, SANS CWE25, OWASPT10 (2004), OWASPT10 (2007), OWASPT10 (2010), WASCTC.	Denim Group y Jeremiah Grossman		2013
NIST ,	NIST, CWE OWASPT10 (2007).	NIST	No	2008
Telligent	OWASP TG v3, WASCTC.	Ninguna	No	2011
Jeremiah Grossman	OWASPT10 (2010), WASC TC (v2).	Ninguna	No	2010
Sectoolmarket	Sectoolmarket, OWASPTGv3, WASCTC.	Sectoolmarket	No	2012

3.4. Aplicaciones Web Vulnerables

Las aplicaciones web vulnerables se pueden agrupar en dos tipos: aquellas que han sido desarrolladas con un propósito específico y aquellas que han sido desarrolladas intencionalmente vulnerables.

En el primer grupo son aplicaciones que tienen alguna utilidad, como blogs, tiendas virtuales o gestores documentales, dentro de las cuales se han detectado vulnerabilidades, una vez puesta en producción la aplicación.

En el segundo grupo están aquellas que se han desarrollado con intención de que sean vulnerables, con el objetivo de probar herramientas de detección o de proporcionar formación a desarrolladores o profesionales de la seguridad informática. Las aplicaciones

web intencionalmente vulnerables pueden ser a su vez de dos tipos: las desarrolladas por organizaciones dedicadas a la seguridad y las desarrolladas por fabricantes de herramientas de detección de vulnerabilidades. De esta forma se tienen tres tipos de aplicaciones vulnerables:

- Aplicaciones **desarrolladas con algún propósito** y que poseen alguna vulnerabilidad conocida.
- Aplicaciones **desarrolladas por los fabricantes** de herramientas de detección de vulnerabilidades web.
- Aplicaciones **desarrolladas por terceros** para comprobar las características de estas herramientas o para impartir formación.

En el primer tipo se encuentran aplicaciones de gestión de contenidos (*Content Management System* (CMS)) como *Joomla* y *WordPress* o aplicaciones desarrolladas a medida. Estas aplicaciones han sido desarrolladas para cumplir con algún propósito; sin embargo, se cometieron errores en alguna de las etapas de su desarrollo, que posteriormente fueron detectados e incluso aprovechados por algún atacante.

En el grupo de aplicaciones vulnerables desarrolladas por los fabricantes de herramientas de detección están las desarrolladas por *Acunetix WVS*, *IBM* o *McAfee*, con las cuales se pueden probar las herramientas de detección que cada uno de ellos desarrolla.

3.4.1. Aplicaciones Desarrolladas por Terceros

El grupo más interesante es el de las aplicaciones vulnerables a propósito desarrolladas por terceros, ya que no están alineadas con las características de ninguna herramienta en particular y suelen incorporar un número relevante de vulnerabilidades.

A continuación se indican las características de las más conocidas y comúnmente utilizadas:

- **WebGoat** [OWA16b]: Cuenta con varios tipos de vulnerabilidades y sigue actualizándose constantemente. Está desarrollada en Java e incluye distintos tipos de tecnologías como JavaScript, XML, etc. Es mantenida como un proyecto de OWASP y se ha desarrollado para poder agregar nuevas vulnerabilidades, si así se desea. Su objetivo es la enseñanza en seguridad y la evaluación de herramientas automáticas. Cuenta con una amplia documentación de las vulnerabilidades presentes en la aplicación, así como vídeo tutoriales.
- **Mutillidae II** [OWA16a]: Cuenta con varios tipos de vulnerabilidades y constantemente se actualiza. Maneja tecnología actual y se le pueden agregar nuevas vulnerabilidades. No posee una temática específica; sin embargo, los ejemplos sí pueden ser considerados como pequeñas aplicaciones reales. Cuenta con distintos niveles de seguridad. Está desarrollado en *Hypertext Preprocessor* (PHP) y tiene como base de datos MySql. Su objetivo es la enseñanza en seguridad web. Cuenta también con una buena documentación de las vulnerabilidades que están en la aplicación y vídeo tutoriales.

- **Damn Vulnerable Web Application (DVWA)** [Ran16]: Cuenta con varios tipos de vulnerabilidades. Maneja tecnología actual y permite agregar nuevas vulnerabilidades. Ha sido pensada para la enseñanza de seguridad web. Al igual que *Mutillidae II* cuenta con niveles de seguridad. Cuenta con una amplia documentación de las vulnerabilidades que están en la aplicación y está desarrollada en PHP y como gestor de base de datos hace uso de MySql.
- **WackoPicko** [Dou16]: Está desarrollada en PHP, simula un portal de ventas de imágenes digitales. Cuenta con varios tipos de vulnerabilidades y hace uso de tecnología actual. No es trivial agregar nuevas vulnerabilidades sin salirse de su funcionalidad. Cuenta con una amplia documentación de las vulnerabilidades que están en la aplicación.
- **The ButterFly Security Project** [Sof16]: Este proyecto tiene como objetivo dar una idea de las vulnerabilidades en aplicaciones web comunes. Está desarrollado en PHP y MySql. Cuenta con varias vulnerabilidades, todas ellas, documentadas.
- **Google Gruyere** [Goo16]: fue una iniciativa de Google, desarrollada en Python, para dar a conocer y enseñar a los desarrolladores las vulnerabilidades que pueden tener las aplicaciones web. Cuenta con amplia documentación de las vulnerabilidades que están presentes en la aplicación, pero cuenta con un conjunto reducido de vulnerabilidades en comparación con las aplicaciones anteriormente explicadas.

En la Tabla 3.5 se indican las principales características de las aplicaciones web vulnerables, pertenecientes a los tres tipos descritos.

Tabla 3.5: Características de las aplicaciones web vulnerables

Aplicación	A propósito	Núm. Vulnerabilidades	Se actualiza	Ampliable	Documentada	Lenguaje	Otros	Desarrollador	Objetivo	Tema
WebGoat	Sí	40	Sí	Sí	Sí	Java	AJAX	OWASP	Enseñanza Evaluación	Ninguno
Mutillidae II	Sí	38	Sí	Sí	Sí	PHP	AJAX	OWASP	Enseñanza	Ninguno
DVWA	Sí	17	Sí	Sí	Sí	PHP		RandomStorm	Enseñanza	Ninguno
WackoPicko	Sí	11	No	No	Sí	PHP		Adam Doupe	Evaluación	Venta de fotos
The ButterFly Security Project	Sí	36	No	Sí	Sí	PHP	AJAX	Pentest Limited	Enseñanza	Venta de discos
Joomla	No	0	Sí	No	No	PHP	AJAX	Joomla	Utilidad	CMS
Wordpress	No	0	Sí	No	No	PHP	AJAX	Wordpress	Utilidad	CMS
Gruyere	Sí	17	No	No	Sí	Python	AJAX	Google	Enseñanza	Ninguno
Acuart	Sí	35	No	No	No	PHP	AJAX Flash	Acunetix	Evaluación	Venta de cuadros
Hackme Bank	Sí	4	No	No	Sí	.NET		McAfee	Evaluación	Banco

3.5. Síntesis del Capítulo

En este capítulo se ha proporcionado una visión general sobre las vulnerabilidades que puedan afectar a las aplicaciones web. Se empezó indicando las diferentes clasificaciones de vulnerabilidades, centrándose especialmente en las que clasifican vulnerabilidades que afectan a las aplicaciones web y describiendo aquellas más conocidas y relevantes. Posteriormente se describieron las relaciones entre las clasificaciones que existen hasta el momento. Finalmente, se realizó una descripción de los distintos tipos de aplicaciones web vulnerables, en especial de aquellas que han sido desarrolladas intencionalmente vulnerables.

Analizando las clasificaciones actuales se observa que ninguna de ellas contiene todos los tipos de vulnerabilidades, o al menos la mayoría. Lo mismo ocurre con las aplicaciones vulnerables para prueba o formación, ya que no existe ninguna que contenga la mayoría de las vulnerabilidades existentes.

Capítulo 4

Análisis de Vulnerabilidades Web

Este capítulo tiene como objetivo introducir las razones que justifican la necesidad del análisis de vulnerabilidades en las aplicaciones web y realizar una descripción panorámica de los distintos tipos de técnicas que se puede usar para realizar este análisis. Primeramente, se comienza describiendo las principales características que hacen que las aplicaciones web necesiten ser analizadas en busca de vulnerabilidades. Seguidamente se exponen las distintas ramas del análisis de seguridad centrándose en las vulnerabilidades web. Posteriormente se exponen las técnicas que se utilizan para realizar los análisis dinámicos de vulnerabilidades en aplicaciones web. El capítulo finaliza con una breve síntesis de lo expuesto en el mismo.

4.1. Necesidad de Análisis de Vulnerabilidades en las Aplicaciones Web

Actualmente el uso de las aplicaciones web se ha extendido a casi cualquier área de la vida diaria, por ejemplo, en el trabajo, en el ocio o en la relación con las administraciones públicas. Estas aplicaciones normalmente están siempre disponibles y a un amplio conjunto de usuarios. Potencialmente todas las personas y máquinas con conexión a Internet pueden en cualquier momento intentar realizar acciones maliciosas sobre estas aplicaciones.

A continuación, se van a describir casos en los que se aprecia de una forma clara y razonada la necesidad de realizar análisis de vulnerabilidad en aplicaciones web.

En muchas ocasiones la información que se muestra en las páginas web es pública, por lo que no necesita protegerse de su robo, pero en otras muchas ocasiones la información que gestiona la aplicación web tiene carácter confidencial y sólo debe ser accedida por el personal autorizado para ello. Su sustracción podría provocar un daño grave, tanto a la organización propietaria, como a terceros.

Existen otras situaciones en las que tanto o más grave que robar la información, es modificarla. Incluso en el caso de que sea información pública, su modificación puede causar graves perjuicios al destinatario.

Otro caso, que se da con relativa frecuencia y que suele causar mayor repercusión mediática, son los ataque contra la disponibilidad de las aplicaciones web. Estos ataques

implican el consumo de muchos recursos materiales y humanos a las organizaciones, que en caso de tener éxito pueden conllevar cuantiosas pérdidas.

Las organizaciones deben mitigar el riesgo de que estas situaciones se den y prevenir, en la medida de lo posible, la detección y explotación de las vulnerabilidades que puedan existir en sus sistemas accesibles desde Internet o redes intranet, que pueda derivar en ataques efectivos. Para ello deben ponerse en el lugar de los posibles atacantes, y buscar las vulnerabilidades que puedan tener.

4.2. Tipos de Análisis de Vulnerabilidades en las Aplicaciones Web

Para la detección de vulnerabilidades en aplicaciones web se pueden realizar dos tipos de análisis: estático y dinámico [ND12] [SM15]. En el primer caso se parte del código fuente de la aplicación y de su descripción funcional proporcionada por el desarrollador. A continuación, de forma manual o utilizando herramientas automáticas, se revisan fragmentos de código en busca de patrones, que puedan corresponderse posteriormente con fuentes de vulnerabilidades cuando el código se esté ejecutando. En el segundo caso se necesita que la aplicación ya esté ejecutándose, de forma que, teniendo la URL y eventualmente unas credenciales válidas, se pueda realizar sobre ella las mismas acciones que podría hacer un usuario, malintencionado o no. Para ello se podrán realizar también pruebas manuales o utilizar herramientas automáticas [MK15].

4.2.1. Análisis Estático

El análisis estático de una aplicación es la revisión de los posibles fallos existentes a un nivel interno, utilizando el código fuente de la aplicación web para buscar vulnerabilidades que puedan permitir un fallo en la seguridad de la aplicación en el momento de su puesta en funcionamiento [HPH+16]. Este tipo de análisis permite la detección y corrección de vulnerabilidades presentes en componentes del código fuente de la aplicación evaluada [FO07]. La tarea de analizar el código en búsqueda de fallos se puede hacer de forma manual o con la ayuda de herramientas de software que lo faciliten, reduciendo el tiempo de análisis. Las herramientas que permiten este tipo de análisis son conocidas como herramientas de prueba de caja blanca (del inglés *White-box Testing*).

Este método cubre casi completamente el código de la aplicación y alcanza la mayoría de los caminos de ejecución posibles. Su desventaja es que la persona que realice el análisis y las herramientas que use tienen que conocer el lenguaje en el que ha sido desarrollada la aplicación. Además, como el código no se ejecuta en ningún momento, suelen producirse muchos falsos positivos.

4.2.1.1. Técnicas de Análisis Estático

El análisis estático de una aplicación puede ser ejecutado de distintas formas, todas ellas con el objetivo de verificar la integridad y funcionalidad de determinadas partes y características propias de la aplicación evaluada. La Figura 4.1 muestra un esquema de esta técnicas de análisis estático.

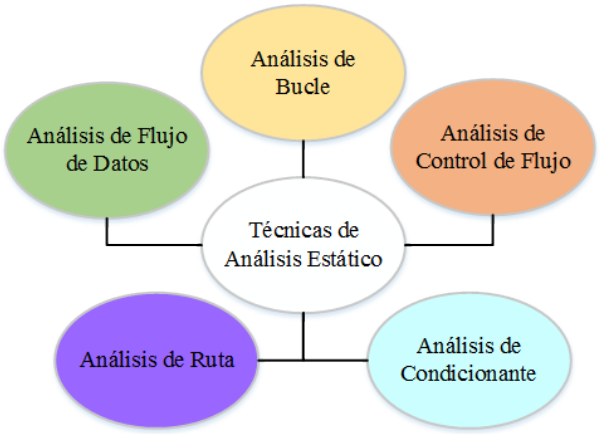


Figura 4.1: Técnicas de análisis estático

Las técnicas más importantes son las siguientes [Kha10] [AW11]:

- **Análisis de Flujo de Datos**: En esta técnica se comprueba el comportamiento de los datos de prueba introducidos para el análisis y se verifica su paso a través de las funcionalidades de la aplicación. Esta técnica puede descubrir fallos como, declaración de variables que no son utilizadas en ningún momento o el uso de variables que no han sido declaradas previamente.
- **Análisis de Control de Flujo**: Esta técnica se utiliza para verificar la estructura de la aplicación y desarrollar casos de prueba en base a esta estructura. Estos casos de prueba son desarrollados para cubrir toda la estructura de control de la aplicación analizada.
- **Análisis de Ruta**: Es una técnica que verifica todos los posibles "caminos" que puede tomar la aplicación en su ejecución con un determinado grupo de datos. Todos los caminos que puede tomar la aplicación deben verificarse al menos una vez. Se usa mucho para verificar aplicaciones altamente complejas.
- **Análisis de Condicionante**: En esta técnica se verifica la implementación de los condicionantes dentro de la aplicación y los datos de prueba que se utilizan en el análisis deben poder verificar cada posibilidad al menos una vez.
- **Análisis de Bucle**: En esta técnica el análisis se enfoca en la correcta implementación de los bucles dentro de la aplicación. Su verificación es sencilla. Existen distintos tipos de bucles: Simple, Anidado y Concatenado.

4.2.2. Análisis Dinámico

El análisis dinámico de vulnerabilidades es un tipo de prueba que verifica el comportamiento de una aplicación ante distintos parámetros y condiciones [RMnGV12]. Este tipo de prueba es realizado por personal o software especializado, conocido como herramientas de análisis de vulnerabilidades o herramientas de prueba de caja negra (*Black-box Testing*). En la Figura 4.2 se puede observar la metodología de ejecución que tiene un análisis dinámico de vulnerabilidades en aplicaciones web, ya sea manual o usando herramientas automáticas.

Figura 4.2: Análisis dinámico

Este tipo de análisis consiste básicamente en enviar peticiones con valores maliciosos en los campos suministrados por el usuario, grabar las respuestas y analizarlas. La aplicación tiene que estar ejecutándose y no depende del lenguaje de desarrollo utilizado. Su principal desventaja es que únicamente analiza las páginas a las que llega, por lo que es necesaria la fase previa de rastreo. Otra desventaja es que no muestra en qué parte del código se encuentra el problema, lo que dificulta su localización.

En este tipo de pruebas no interesa conocer el mecanismo de funcionamiento interno de la aplicación web que analiza. El análisis se realiza sin conocer de antemano la arquitectura, el lenguaje de desarrollo o la plataforma sobre la que se ejecuta la aplicación web [BBGM10] [ND12] [SM15]. Para el análisis dinámico lo único que interesa es analizar las respuestas de la aplicación web a las solicitudes maliciosas enviadas por el usuario que está llevando a cabo el análisis.

4.2.2.1. Fases del Análisis Dinámico

Un análisis dinámico de vulnerabilidades se divide principalmente en dos fases, que se representan en la Figura 4.2. Una primera fase pasiva, también llamada de rastreo, y una segunda activa. En la primera fase se pretende localizar el mayor número posible de elementos de los que componen la aplicación. El objetivo es localizar todos los directorios y archivos de la aplicación, y los puntos de entrada de cada uno de ellos (por ejemplo, cabeceras, campos de formularios y cookies).

En la segunda fase se realizan determinadas pruebas sobre los componentes de la aplicación para encontrar vulnerabilidades como las que figuran en las listas antes mencionadas. Se realizarán una serie de pruebas sobre estos elementos para comprobar por ejemplo la fortaleza del método de autenticación, o si filtra correctamente la información que introduzca un usuario. Este análisis dinámico se puede realizar de forma manual,

siguiendo alguna guía como las de OWASP, utilizando una herramienta de análisis automático de vulnerabilidades como las que se indican en el apartado siguiente, o una combinación de ambos.

4.3. Herramientas de Análisis Dinámico de Aplicaciones Web

Las herramientas automáticas de detección de vulnerabilidades se han hecho populares como apoyo a los profesionales en la búsqueda de vulnerabilidades en aplicaciones web. Estas herramientas analizan de manera automática o semiautomática la aplicación web, siguiendo las fases indicadas de rastreo y prueba, realizando ataques maliciosos en busca de vulnerabilidades que puedan ponerla en riesgo.

Para que la herramienta pueda iniciar el análisis, se le debe proporcionar su URL o URLs y eventualmente las credenciales necesarias. Estas herramientas suelen contar con 3 módulos principales:

1. Un rastreador, que se encarga de iniciar el rastreo de la aplicación partiendo de las URLs proporcionadas, recupera las páginas y archivos accesibles de ésta e identifica todas las entradas que pueden encontrarse en formularios, al igual que los parámetros que se envían al servidor de la aplicación.

2. Un módulo de ataque que se encarga de generar valores que pueden explotar algún tipo de vulnerabilidad. Estos valores son generados para cada entrada que ha encontrado el rastreador y para cada tipo de vulnerabilidad.

3. Un módulo de análisis, el cual se encarga de analizar las respuestas retornadas por el servidor o la aplicación de los distintos ataques en busca de algún patrón que pudiera dar como válido el ataque realizado, o para retro-alimentar a los otros módulos.

El primer módulo realiza la fase de rastreo o etapa pasiva, y el segundo y el tercero la fase de prueba o etapa activa.

4.3.1. Etapa Pasiva

La etapa pasiva del análisis tiene como tarea navegar a través de la aplicación web, con el objetivo de localizar todas las páginas, enlaces, formularios y también identificar todos los vectores de entrada asociados a ellos [PZK15].

En esta etapa se ejecutan dos tareas:

- **Rastreo**: La herramienta busca todos los posibles enlaces y directorios que componen la aplicación web, con el fin de adquirir su código HTML. Páginas ocultas o protegidas dentro de la aplicación web dificultan esta tarea y pueden incidir en el análisis que se realiza posteriormente. Por ello es necesario que en la configuración del escáner se prevea esta posibilidad, ingresando los parámetros que la herramienta debería utilizar en el caso de no poder acceder a un enlace o formulario por falta de

credenciales o parámetros específicos. Al finalizar esta etapa, el escáner debería ya contar con toda la estructura de la aplicación web en formato HTML [The12].

- **Identificador de Parámetros de entrada**: En esta tarea el escáner realiza ingeniería inversa sobre el código HTML para identificar formularios y campos de entrada de datos, así como otra información que pueda intercambiar el cliente y el servidor como son las cabeceras y las *cookies*.

En la Figura 4.3 se muestra una captura de pantalla de la herramienta Acunetix WVS durante la etapa de rastreo de la aplicación DVWA. Se puede observar la creación de la estructura de ficheros que compone la aplicación web y en la parte inferior se aprecia la navegación que realiza dentro de la aplicación web.

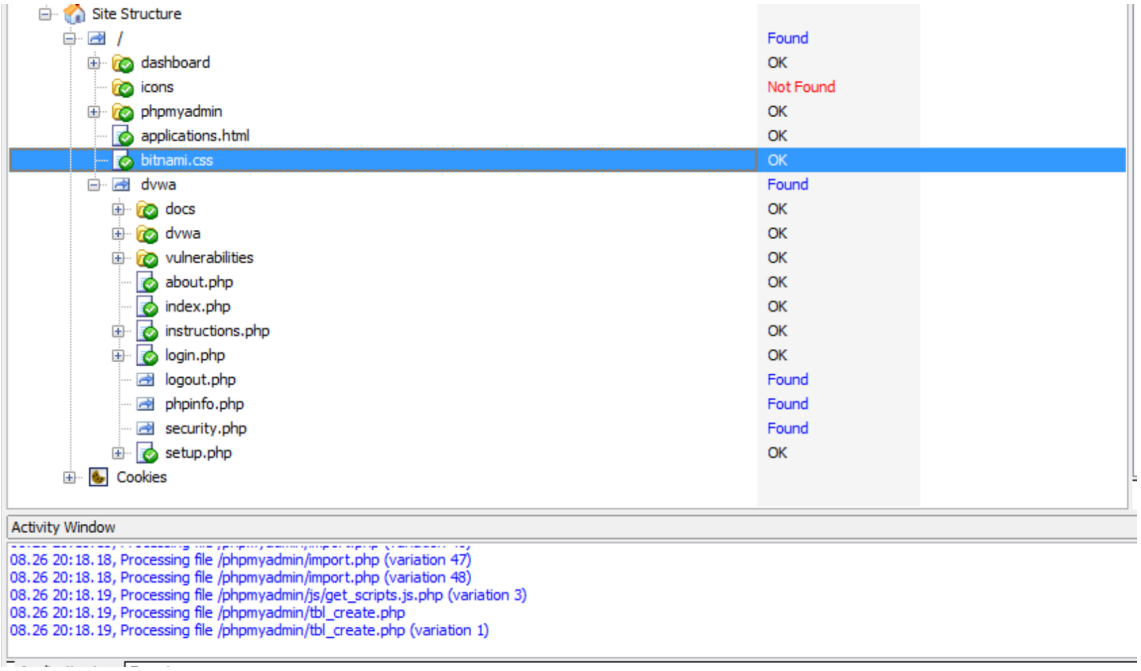


Figura 4.3: Etapa de rastreo del sitio DVWA con Acunetix WVS

4.3.2. Etapa Activa

La etapa activa del análisis de un escáner de vulnerabilidades web se ejecuta una vez que el escáner ha identificado la estructura y todos los parámetros de entrada de la aplicación. La herramienta inicia el ataque hacia la aplicación, el cual consiste en la inyección de valores en cada parámetro, para posteriormente analizar las respuestas que recibe por parte de la aplicación frente a lo enviado [PZK15]. En las Figuras 4.4 y 4.5 se puede observar el comportamiento de Acunetix WVS durante la etapa activa del análisis de la aplicación web DVWA.

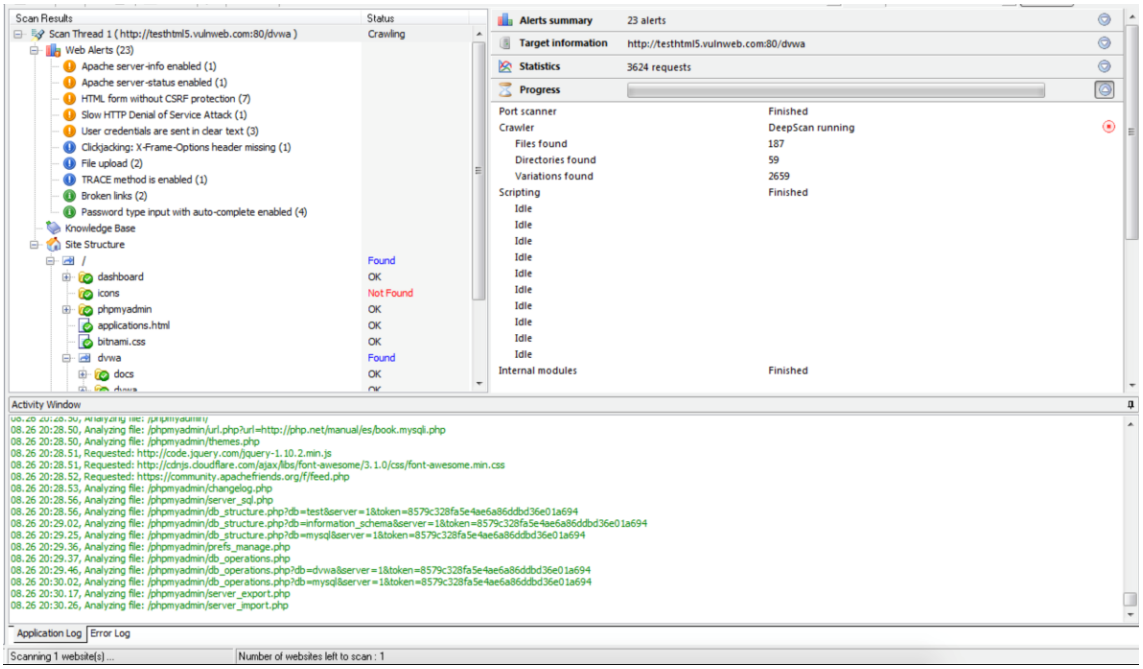


Figura 4.4: Etapa activa del análisis de DVWA con Acunetix WVS

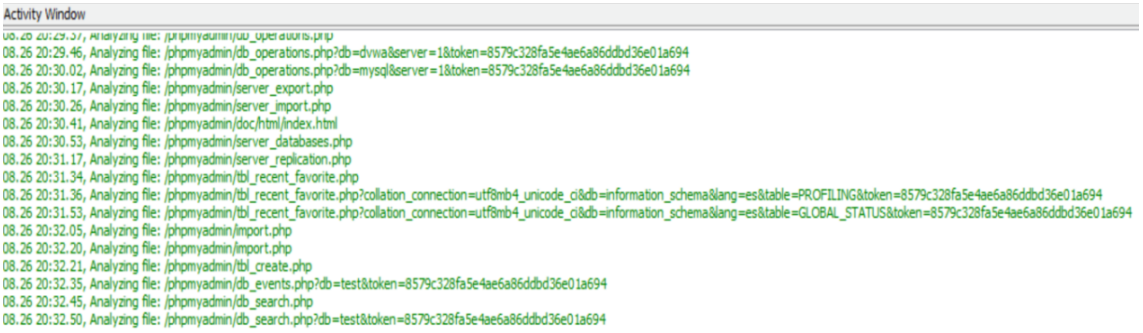


Figura 4.5: Análisis de DVWA con Acunetix WVS - Análisis de las respuestas de DVWA

Un factor determinante para el óptimo funcionamiento de un escáner dinámico es la capacidad de realizar, en la fase anterior de rastreo, un reconocimiento completo de todas las páginas, enlaces y campos de entrada de la aplicación lo más profundo posible para que después se pueda analizar todo sin dejar ninguna página, enlace o campo de lado.

La etapa activa del análisis se completa en dos tareas:

1. **Ataque**: En esta tarea el escáner genera datos maliciosos para realizar ataques a la aplicación utilizando los parámetros localizados en la etapa anterior. Generados estos datos, el escáner los envía y espera por la respuesta de la aplicación web para analizarla [The12] en busca de evidencias que confirmen si el ataque ha tenido éxito.

2. **Análisis de respuestas**: En esta tarea el escáner requiere ejecutar ingeniería inversa sobre cada una de las respuestas de la aplicación web, en busca de información válida, según el criterio de la herramienta, para generar el informe final. El escáner toma la decisión de considerar si existe una vulnerabilidad o no cuando, al analizar una respuesta de la aplicación web, ésta contiene un grupo de caracteres que coincide con alguno de los mensajes que el escáner mantiene en su base de conocimiento. Esta etapa es la que representa una mayor cantidad de esfuerzo por parte del escáner, ya que las respuestas que son generadas por la aplicación web están elaboradas para ser consumidas por seres humanos.

Por ejemplo, si busca una vulnerabilidad del tipo XSS, podrá enviar el código presentado en el Algoritmo 2 como valor de un campo de formulario (Figura 4.6) y revisará la respuesta en busca de ese código (Algoritmo 3).

Algoritmo 2: Script para encontrar una vulnerabilidad XSS

```
1  <script>Alert(123)</script>
```

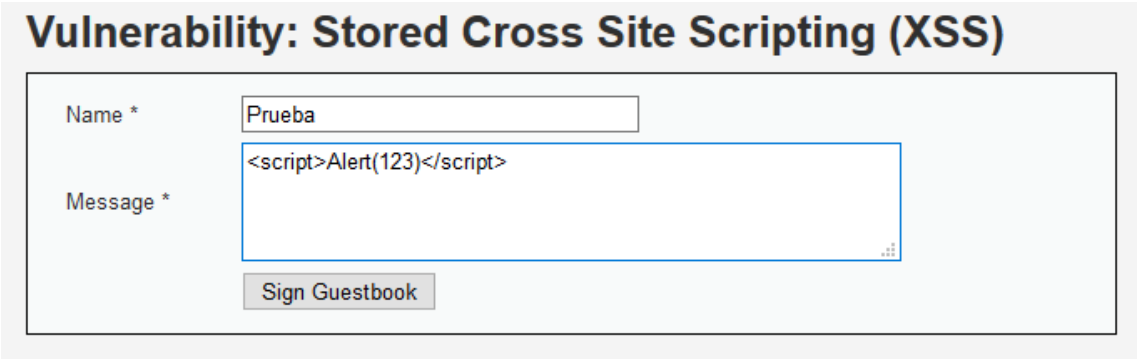


Figura 4.6: Prueba de XSS en un campo de formulario de DVWA

.

Algoritmo 3: Fragmento de código en DVWA que confirma un XSS

```
1  <div id="guestbook_comments">
2      Name: Prueba<br />
3      Message: <script>alert(123)</script><br />
4  </div>
```

Aunque se confirma visualmente la existencia de la vulnerabilidad del tipo XSS (Figura 4.7), la herramienta automática sólo puede deducirlo revisando el código HTML de respuesta.

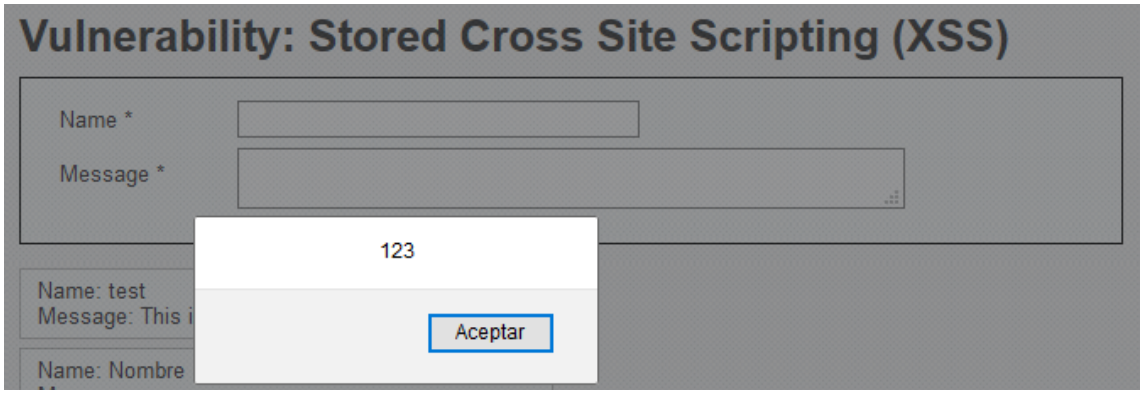


Figura 4.7: Confirmación de XSS en DVWA

4.3.3. Tipos de Herramientas de Análisis Dinámico

Existe una gran cantidad de herramientas de análisis dinámico para búsqueda de vulnerabilidades en aplicaciones web. Estas herramientas se pueden dividir en dos grupos: gratuitas y comerciales. En cada uno de estos grupos existen además herramientas:

- **Especializadas**: Estas herramientas están diseñadas para buscar determinadas vulnerabilidades. Cuentan con una base de conocimiento específico que les permite generar parámetros de entrada para ataques de vulnerabilidades determinadas y, posteriormente, buscar patrones de respuesta concretos de aquellas vulnerabilidades para las cuales fueron concebidas. Por ejemplo, vulnerabilidades de tipo SQLI o XSS únicamente.

- **Genéricas**: Estas herramientas están diseñadas para detectar la mayoría de vulnerabilidades conocidas y además pueden ofrecer la capacidad de realizar búsquedas específicas. Este tipo de herramientas cuentan con características más amplias que las que presenta una herramienta especializada. Tienen una base de conocimiento mucho más grande que las otras, lo que les permite hallar una mayor cantidad de vulnerabilidades de distintas clases.

4.3.4. Principales Debilidades

El uso de herramientas de penetración y análisis dinámico de aplicaciones permite reducir tiempo y esfuerzo dentro del ciclo de desarrollo de una aplicación y además permite enfocar mayores esfuerzos en tareas de seguridad más complejas. Este tipo de herramientas no son sencillas de configurar para quien no está familiarizado con ellas [Bar11].

Para que estas herramientas puedan encontrar vulnerabilidades como XSS, inyección SQL u otros tipos de inyección [BFOG08] [OWA14], es necesario realizar un rastreo inicial lo más completo posible de la aplicación web [Bar11]. Las herramientas actuales en general no son capaces de realizar este rastreo completo [BBGM10] [DCV10], principalmente por dos motivos:

1. Estas herramientas no son capaces de rellenar adecuadamente los formularios, especialmente los que tienen restricciones muy fuertes en sus campos. Por ejemplo, los campos de un tipo y tamaño determinados, o campos cuyo valor depende del que se haya puesto en otro campo anterior. En la Figura 4.8 se ve un ejemplo de esta situación. Para que se pueda avanzar en el flujo definido en la aplicación y llegar a otras fases del proceso, es necesario que en cada campo se introduzca un valor adecuado. Estos valores deberán cumplir ciertos requisitos. Primero deben ser del tipo esperado por el campo. Por ejemplo, si un campo se llama *E-mail*, se necesitará un valor similar a una dirección de correo electrónico, o si se llama *Telephone* tendrá que ser un número que pueda servir como número de teléfono. Segundo se deberán tener en cuenta las relaciones entre campos. Por ejemplo, el valor de un campo "Localidad" deberá ser consecuente con el valor de un campo anterior "Provincia".

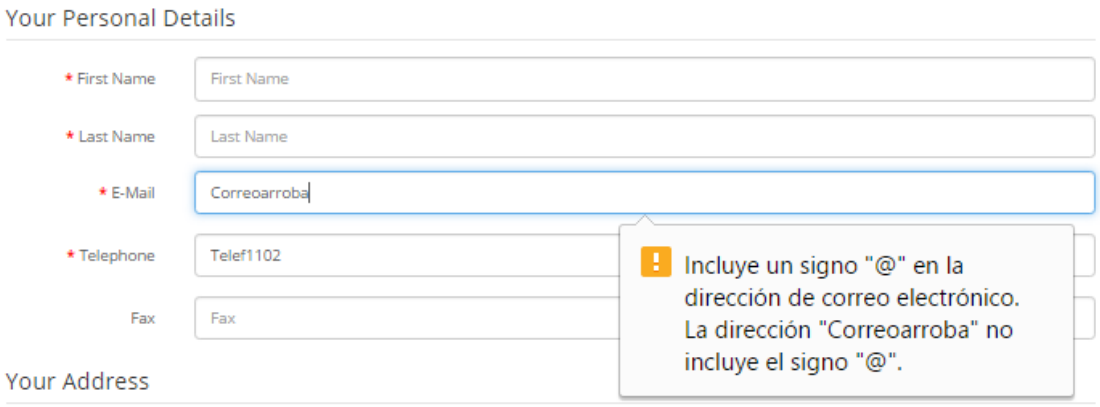


Figura 4.8: Ejemplo de formulario de alta de usuario

2. A menudo no ejecutan correctamente el código de cliente que envía la aplicación al navegador, principalmente el código JavaScript o las peticiones AJAX. Este tipo de código se carga junto con el código HTML y, ante ciertas acciones del usuario o ciertas situaciones, modifica el comportamiento de la página visitada sin necesidad de la interacción con el servidor. Las herramientas automáticas de detección simulan un navegador web para cargar este código y replicar el comportamiento de un usuario, pero en muchos casos el comportamiento no es el esperado.

4.3.4.1. Rellenar Campos con Valores Válidos

Los formularios web están formados por campos, en los que se introduce o selecciona una serie de valores, que son enviados al servidor web para su proceso. Estos campos pueden ser básicamente de los siguientes tipos:

- Oculto o visible al usuario.
- Obligatorio u opcional.
- De selección (única, múltiple, combos, botones de selección, etc.) o de texto (libre o clasificado).
- Con sus valores dependientes de los valores de otros o independientes.

Una vez que los valores llegan al servidor, éste los procesa y devuelve una respuesta distinta en función de si los valores son válidos o no. Devolverá una página (la misma u otra) con distinto contenido, si los valores han tenido éxito; y devolverá el mismo formulario que habrá que completar correctamente o un mensaje de error, si los valores son inválidos o no son suficientes.

Las soluciones para encontrar las páginas que se encuentran detrás de los formularios son dos principalmente. La primera opción es que la herramienta navegue por la aplicación recopilando todos los enlaces que se encuentren, e introduciendo valores prefijados en los campos de los formularios. En este caso se suele complementar con la selección automática de valores para los campos de selección (lista de opciones, botones, o cajas) por parte de la herramienta. Por ejemplo, en la Figura 4.9 se ve la pantalla de una de las herramientas en las que se indican, previamente al rastreo, los valores a introducir en cada campo. Para los casos en los que se encuentre con un campo no contemplado, las herramientas tienen la opción de introducir un valor por defecto.

La segunda solución consiste en guardar la navegación que realice un usuario de la aplicación. Después, en una fase posterior de rastreo automático, se utilizarán los valores que haya introducido el usuario en los campos de los formularios. El principal inconveniente de este tipo de rastreo de formularios es que se confía plenamente en que el usuario recorra la aplicación entera. Durante su navegación el usuario debe introducir valores que vayan a ser correctos posteriormente en el rastreo automático que haga la herramienta. Adicionalmente tiene el problema de que no suele disponer de un método efectivo para determinar cuándo los valores introducidos en una página han tenido éxito o no.

Un caso particular de formulario son los de inicio de sesión. En estos formularios el usuario introduce generalmente un identificador y una contraseña que la aplicación debe validar. Las herramientas actuales de análisis de vulnerabilidades suelen abordar este formulario de forma distinta al resto de formularios. Para ello permiten al usuario de la herramienta por ejemplo grabar una macro con el inicio de sesión, que posteriormente usará cuando sea necesario.

Las principales características de los escáneres de uso habitual en lo referente al relleno automático de formularios son las siguientes:

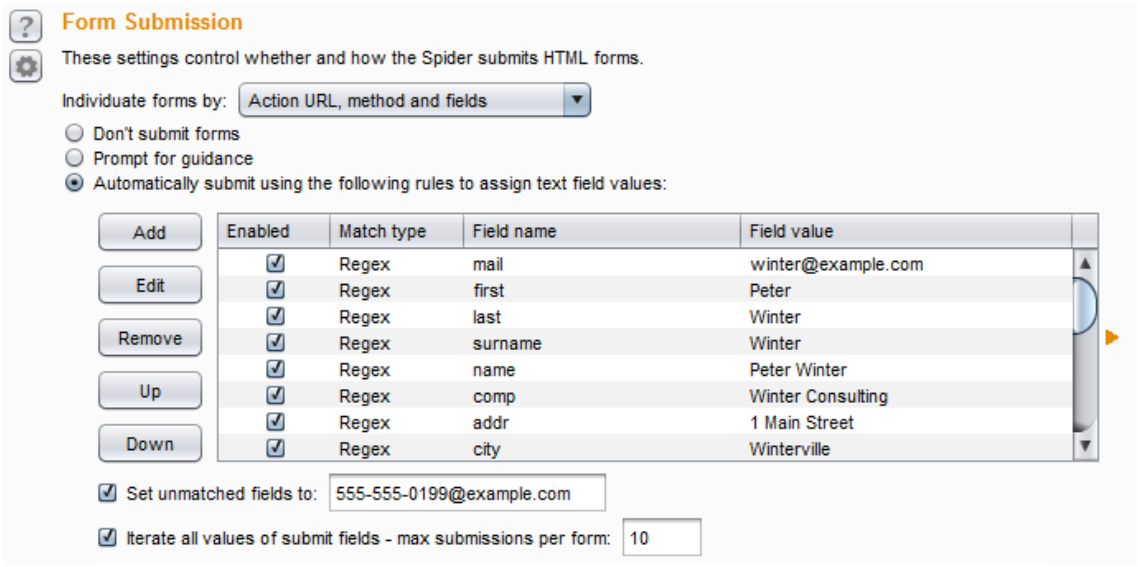


Figura 4.9: Valores prefijados para los campos en Burp Suite

- **Grabar navegación**: la herramienta puede grabar una navegación de usuario y usar los valores utilizados por el usuario en el rastreo posterior.
- **Solicitar datos durante el rastreo**: si así se le indica, la herramienta pedirá durante el análisis la introducción de los valores de los campos de los formularios.
- **Definir valores**: El usuario puede configurar la herramienta con valores asociados a campos, que posteriormente se usarán.
- **Valor por defecto**: La herramienta permite definir qué valor poner en los campos, cuando no se encuentra ninguno que concuerde con ese nombre de campo.
- **Valores por URL**: Permite, dependiendo de la URL a analizar, definir unos valores específicos para los campos de los formularios que localice.
- **Definir el campo**: La herramienta permite usar "Wildcards" para el nombre de los campos.
- **Definir el valor**: La herramienta permite usar "Wildcards" para el valor de los campos.
- **Inicio de sesión**: la herramienta permite grabar previamente una macro de inicio de sesión.

4.3.4.2. Web Oculta

Un caso particular del problema de obtener valores válidos para los formularios se da en la exploración de la web oculta o profunda (*Deep Web Crawling*). El objetivo del rastreo

de la web oculta es obtener la información que está detrás de formularios de búsqueda y eventualmente incorporarla a los buscadores tradicionales.

Hay dos formas principalmente para atacar la web oculta [Wri08]. Se pueden explorar a priori los formularios de búsqueda [CHM11] o se puede realizar la búsqueda en tiempo real. En el primer caso los datos obtenidos se indexarán en el buscador para usarlos posteriormente. Como consecuencia, puede que los datos que se muestren no sean actuales. En el segundo caso se realiza la búsqueda sobre un formulario que redirige la consulta a varios formularios predefinidos, según el dominio que se consulte. Este tipo de solución sólo será posible para un conjunto finito de dominios.

En el primer caso habrá que obtener los valores que poner en cada campo. Se pueden obtener de distintas fuentes [MKK+08] [XTH08]: los que aparezcan en los campos de selección; mantener una serie de valores para campos habituales como meses, días de la semana, códigos postales, etc.; extraerlos de las páginas de respuesta usando la medida *Term Frequency-Inverse Document Frequency* (TF-IDF), o una similar; u obtenerlos de fuentes de información adicionales. Una vez que se tiene una base de datos de campos y valores, se puede añadir a cada par campo-valor un peso según el éxito que haya tenido cuando se ha usado.

Adicionalmente se puede definir un método para saber si los campos enviados proporcionan respuestas suficientemente distintas, de forma que no haya que probar con todo el producto cartesiano de campos [MKK+08].

4.4. Características de Herramientas Representativas

Una vez vistas las características generales de las herramientas de análisis de vulnerabilidades web, se describen las características concretas de algunas de las herramientas utilizadas habitualmente en estos análisis de seguridad. Se han elegido dos herramientas gratuitas y otras dos comerciales. Todas ellas se encuentran entre las más utilizadas y también son las que se suelen utilizar en los trabajos que comparan este tipo de herramientas.

4.4.1. OWASP ZAP

OWASP Zed Attack Proxy (OWASP ZAP) [The16] es una herramienta gratuita de análisis dinámico de vulnerabilidades. En la Figura 4.10 se muestra la interfaz de usuario de OWASP ZAP.

OWASP ZAP forma parte del grupo de proyectos de la fundación OWASP. Es ampliamente utilizado alrededor del mundo y la comunidad de voluntarios que la mantiene y desarrolla ofrece gran cantidad de documentación y soporte [The12]. Es considerado uno de los escáneres de vulnerabilidades web gratuito más populares del mundo [The12]. Su diseño y configuración permite ser usado tanto por usuarios expertos, con un alto nivel de conocimiento de este tipo de herramientas, como por desarrolladores o usuarios en general, que no están familiarizados o son nuevos en la realización de pruebas de penetración en aplicaciones web. Puede realizar distintos tipos de análisis y ataques al configurar perfiles

Figura 4.10: Captura de pantalla de la interfaz de usuario de la herramienta OWASP ZAP

específicos para ajustarlos a las características de la aplicación que se desea analizar. Es utilizado como un escáner automático, en su conjunto, pero también presenta una serie de herramientas individuales que facilitan la búsqueda de vulnerabilidades de forma manual [The12].

Con OWASP ZAP se pueden realizar los siguientes dos tipos de análisis de forma independiente o conjunta:

- **Rastreo activo**: Es el modo más utilizado por quienes emplean esta herramienta. El rastreo activo aplica políticas de búsqueda que pueden ser configuradas por el usuario o puede utilizar sencillamente la política por defecto que hace un análisis completo de la aplicación web en busca de vulnerabilidades.
- **Rastreo Pasivo**: En este modo de análisis OWASP ZAP etiqueta de manera automática aquellas respuestas por parte de la aplicación web en base a un grupo de reglas específicas del usuario. La herramienta no realiza ningún análisis invasivo.

Posee las siguientes características:

- **Complementos**: OWASP ZAP soporta la instalación de complementos para agregar mayor funcionalidad a la herramienta.
- **Alertas**: Diferenciación de alertas de vulnerabilidades mediante el uso de gráficos con colores que determinan los niveles de gravedad de cada vulnerabilidad hallada (Alta, Media, Baja, Información, Falso Positivo)
- **Anti CSRF *Tokens***: Permite la detección del uso de *tokens* en el envío de formularios, aumentando de esta forma la dificultad de realizar ataques CSRF. Con el uso de esta funcionalidad OWASP ZAP puede identificar estos *tokens* y almacenarlos conjuntamente con la URL que lo generó, para usarlo en ataques de este tipo.

- ***Application Programming Interface* (API)**: OWASP ZAP provee de un API a través de la cual se puede interactuar con la herramienta mediante programación. Soporta los formatos *Javascript Object Notation* (JSON), HTML y XML. Con el uso de esta API se puede acceder a funcionalidades principales como rastreo activo y *Spider*.
- **Autenticación**: Soporta distintos métodos de autenticación para el acceso hacia las aplicaciones web objeto de análisis, a saber: Método Manual, permite al usuario ingresar a la aplicación mediante la introducción de sus credenciales durante la fase de captura y navegación proxy; Basada en Formulario, este método es usado cuando la autenticación se realiza mediante el envío de un *script* a través de una petición GET, el script debe contener el par credencial de Usuario/Contraseña otorgado por el usuario.
- ***Break Points***: Con esta funcionalidad OWASP ZAP puede interceptar todas las peticiones y respuestas para posteriormente modificarlas. Con esta funcionalidad se puede realizar un *bypass* en la validación del usuario en la aplicación y obtener sus credenciales.
- ***Contexts***: Permite asociar diferentes tipos de URLs con un mismo sitio.
- ***Data Driven Content***: Con la utilización de *Contexts* se reduce el número de peticiones en una misma página de un sitio.
- **Filtros**: Puede realizar filtros sobre cada petición y respuesta durante el reconocimiento del sitio utilizando la función de Proxy.
- **Exclusión de URLs (forma global)**: Con el uso de expresiones regulares ZAP ignora las URLs que coincidan con estas expresiones dentro de la aplicación durante todo el proceso de análisis.
- **Sesiones HTTP**: Almacena y utiliza distintas sesiones mediante *cookies* dentro de una misma aplicación, forzando a que todas las peticiones se hagan en una sesión particular sin destruir ninguna otra.
- **Proxy Interceptor**: Funcionalidad principal de la herramienta que permite ver y almacenar en un registro todas las peticiones que se hacen hacia una aplicación web y sus respectivas respuestas.
- **Interfaz**: OWASP ZAP permite cambiar el aspecto de su interfaz para cumplir determinados roles y facilitar el acceso hacia funcionalidades propias de cada rol (Modo Seguro, Modo Protegido, Modo Estándar, Modo Ataque).
- **Reglas**: OWASP ZAP soporta reglas para análisis activo o pasivo. Todas las reglas que utiliza ZAP son complementos propios de la herramienta y su actualización es sencilla a través del administrador de complementos.

- **Alcance**: OWASP ZAP permite determinar el alcance de búsqueda en la aplicación, utilizando el grupo de URLs relacionadas en la característica de contexto. Por defecto, el alcance es nulo, es decir, no tiene límite.

4.4.2. Arachni

Es una herramienta gratuita de código abierto de análisis dinámico de vulnerabilidades en aplicaciones web. Está desarrollada en *Ruby*. Presenta una interfaz web que permite una fácil configuración. Esta característica lo convierte en un escáner multiplataforma (Windows, Linux y Mac OS X) [Ara16]. En la Figura 4.11 se puede observar la interfaz gráfica que presenta Arachni.

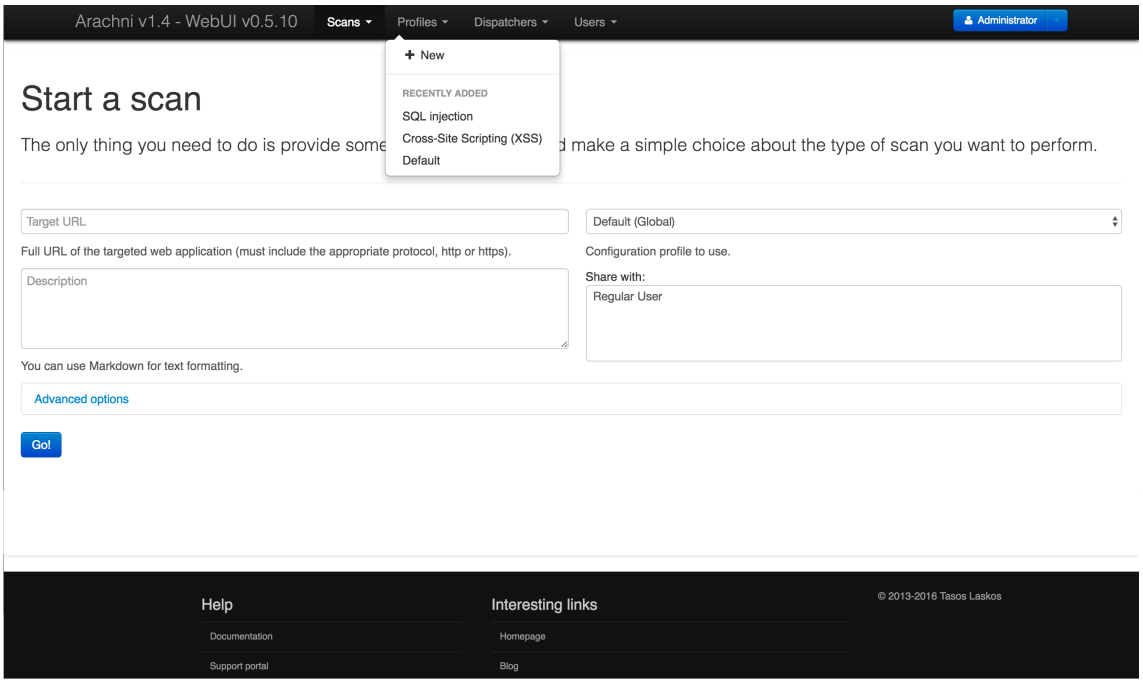


Figura 4.11: Captura de pantalla de la interfaz gráfica de Arachni

Adicionalmente, es capaz de realizar análisis de aplicaciones web en colaboración con distintas instancias o "*dispatchers*" de la aplicación, que funcionan como uno solo, para reducir el tiempo de rastreo y análisis. Arachni puede ampliarse y mejorar en funcionalidades. Esto gracias a que es una herramienta de código abierto.

Sus primeras versiones no presentaban una interfaz gráfica y por lo tanto la configuración y análisis de una aplicación se realizaba utilizando la ventana de comandos. En la Figura 4.12 se puede observar la presentación en la ventana de línea de comandos para ejecutar Arachni.

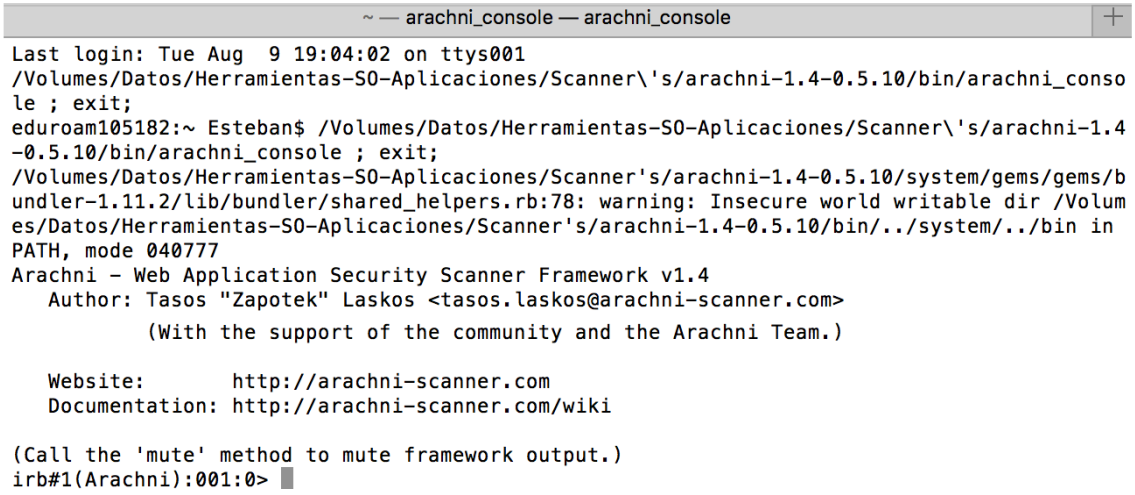


Figura 4.12: Captura de pantalla de línea de comandos de Arachni

Arachni cuenta con una estructura altamente configurable y posee distintos complementos específicamente diseñados para ambas etapas del análisis: pasiva y activa.

Arachni puede ejecutar tres distintos tipos de análisis:

- **Análisis Directo**: Con la configuración establecida y con la URL de la aplicación que se desea analizar, se realiza un análisis completo sin ninguna otra configuración adicional.
- **Análisis Remoto**: Mediante el uso de instancias o "*dispatchers*" se puede asignar a ellos la tarea de análisis de aplicaciones web sin que el tráfico se genere desde el ordenador de origen.
- **Análisis en Distribuido o "*Grid*"**: Con un grupo de instancias o "*dispatchers*" de Arachni, se puede distribuir la carga de trabajo del análisis de aplicaciones web a lo largo de la red de instancias y acelerar los procesos de rastreo y análisis en general.

Las principales características de Arachni son:

- **Autenticación**: Arachni permite la autenticación a la aplicación web basada en *cookies*, *Secure Sockets Layer* (SSL), formulario, *NT Lan Manager* (NTLM) y Kerberos entre otros.
- **Detección Automática de Desconexión**: Arachni detecta automáticamente el método de desconexión del usuario de la aplicación web y permite además la reconexión de la sesión para continuar con el análisis sin interrupciones.
- **Abstracción de Interfaz**: Arachni puede ser utilizado directamente por el usuario mediante dos interfaces: Interfaz de ventana de línea de comandos, Interfaz Web.
- **Parámetros de Entrada**: Arachni permite la utilización de parámetros de entrada configurados por el usuario para cada análisis. Utiliza un formato básico de

clave/valor para determinar el nombre del campo y el valor que se otorga en ese campo en cada formulario.

- **Características ofrecidas para el desarrollador**: Arachni cuenta con características que permiten a desarrolladores crear aplicaciones que se puedan comunicar con Arachni y también controlar Arachni. Para ello Arachni cuenta con las siguientes APIs que facilitan el trabajo:
  - *API Representational State Transfer (REST)*: Mediante esta API los desarrolladores pueden interactuar con Arachni. El formato de respuestas pueden ser objetos JSON o XML.
  - *API Remote Procedure Call (RPC)*: Esta API facilita a los desarrolladores las tareas de comunicación con la herramienta y la administración de sus funcionalidades en modo distribuido.

4.4.3. Acunetix WVS

Acunetix WVS [Acu15] es una herramienta comercial especializada en auditar aplicaciones web y buscar vulnerabilidades y fallos que puedan comprometer la integridad de la aplicación y la información que contiene. En la Figura 4.13 se puede observar la interfaz de usuario que presenta esta herramienta.

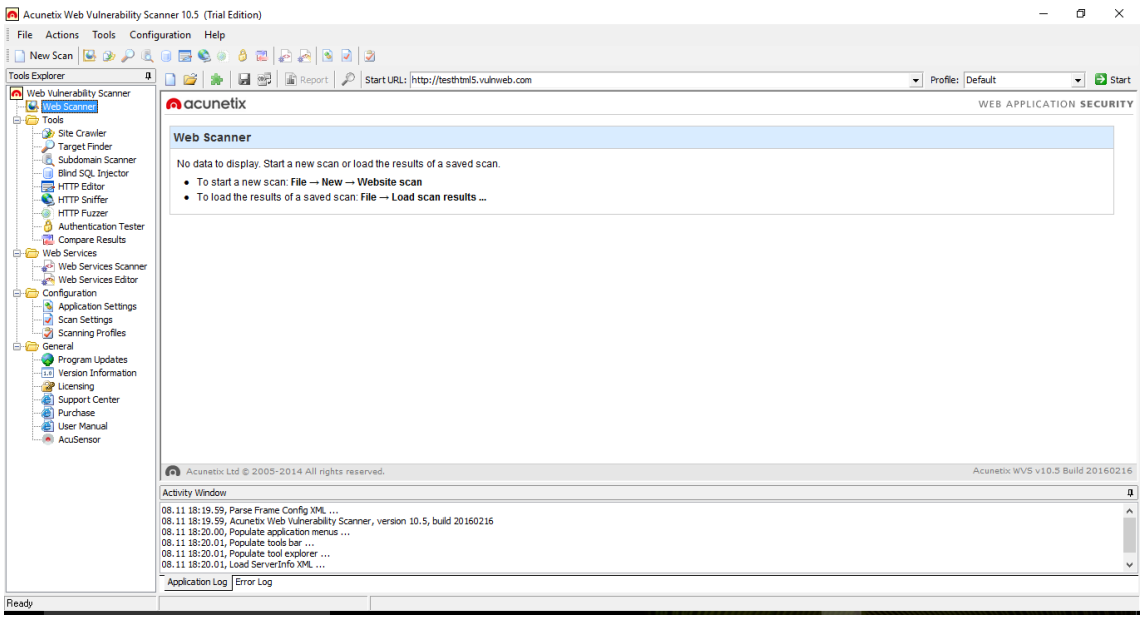


Figura 4.13: Captura de pantalla de la interfaz gráfica de Acunetix WVS

Acunetix WVS es muy popular gracias a la gran cantidad de características que ofrece permitiendo configurar la herramienta para aumentar la precisión del análisis y así obtener resultados más fiables en su informe. En su página web ofrece además el servicio gratuito de análisis de vulnerabilidades de aplicaciones que se encuentren en línea.

Las características y funciones más destacadas de Acunetix WVS son:

- **Web Scanner**: Realiza una auditoría automática de un aplicación web, consistente en dos etapas:
  - *Crawiling*: Utilizando *Acunetix DeepScan*, *Acunetix WVS* navega automáticamente dentro de la aplicación, en busca de enlaces y páginas con el fin de establecer una estructura lo más precisa posible de la aplicación que se está analizando. Este proceso enumera todos los archivos dentro de la aplicación y es fundamental para asegurar que todos estos sean analizados.
  - *Scanning*: Con *Acunetix WVS* se ejecuta la búsqueda de vulnerabilidades web dentro de cada fichero presente en la aplicación, simulando el ataque de un *hacker*. El resultado es procesado y se muestra en los informes que se generan.
- **AcuSensor Technology**: Esta tecnología es única de *Acunetix WVS* y permite identificar una mayor cantidad de vulnerabilidades que cualquier análisis dinámico tradicional. Está diseñado para reducir cualquier falso positivo. Esta mejora en la precisión de búsqueda se produce al combinar las técnicas de análisis dinámico con el análisis de código al tiempo en que éste se ejecuta. Para lograr esto, es necesario que se instale un agente sensor en la aplicación web permitiendo así la comunicación entre el agente sensor y *Acunetix WVS*. De momento es únicamente posible en aplicaciones desarrolladas en PHP y .NET.
- **PortScanner**: *Acunetix WVS* puede realizar un rastreo en los puertos abiertos del servidor web en el cual se encuentra la aplicación que se desea analizar, para revisar la seguridad del servicio que se ejecuta en el puerto. El usuario puede además configurar sus propios servicios de revisión de seguridad en red mediante un *script*.
- **Target Finder**: Sirve para que *Acunetix WVS* pueda localizar servidores web (puertos 80 y 443) dentro de un rango IP dado. De cada servidor hallado se muestra la cabecera de respuesta y el software del servidor. Los puertos de búsqueda son configurables.
- **Subdomain Scanner**: Identifica los subdominios activos dentro de un dominio.
- **Blind SQL Injector**: Es una funcionalidad para la extracción de datos de una base de datos. Mediante el uso de técnicas de inyección a ciegas de SQL (*Blind SQL Injection* (BSQLI)) enumera base de datos y tablas, descarga los datos y también lee archivos específicos de sistema del servidor web. Además, permite realizar pruebas manuales de tipo inyección SQL sobre vulnerabilidades descubiertas en un análisis.
- **HTTP Editor**: Permite la creación, análisis y edición de las peticiones HTTP y también de las respuestas del servidor. Puede codificar y decodificar texto y URLs hacia distintos formatos.

- **HTTP Sniffer**: Actúa como un proxy que captura, examina y modifica el tráfico HTTP entre un cliente y el servidor. La información capturada se puede importar a la sesión de análisis de la aplicación y servir de guía para el rastreo automático de la misma. Para activar esta funcionalidad en *Acunetix WVS* es necesario previamente configurar en el navegador web a *Acunetix WVS* como proxy HTTP.
- **Authentication Tester**: Permite ejecutar ataques hacia las páginas de autenticación de las aplicaciones, utilizando diccionarios predefinidos. Los diccionarios pueden ser modificados y agregar nombres de usuario y contraseñas personalizadas.
- **Web Services Scanner and Web Services Editor**: Puede ejecutar un análisis automático de vulnerabilidades sobre servicios web (del inglés *Web Services Description Language* (WSDL)). El editor puede importar un WSDL local o en línea para editar y ejecutar operaciones de servicios web sobre distintos puertos y obtener un análisis profundo de las respuestas y solicitudes WSDL.
- **Acunetix Web Vulnerability Scanner *Software Development Kit* (SDK)**: Se puede desarrollar características adicionales para el análisis que hace *Acunetix WVS*. Estas características deben ser escritas en JavaScript y requiere de la instalación de las herramientas para el desarrollo propio de *Acunetix WVS*.
- **Informes**: Los informes en *Acunetix WVS* son generados por la aplicación *Acunetix WVS Reporter*. Esta aplicación se ejecuta después de haberse completado el rastreo o desde la aplicación principal de *Acunetix WVS* y admite la generación de distintos tipos de informes incluyendo informes de desarrollo, informes ejecutivos, informes completos estándar e informes comparativos de dos rastreos distintos.

4.4.4. HP WebInspect

Es una herramienta desarrollada por HP para el análisis dinámico de vulnerabilidades en aplicaciones web. En la Figura 4.14 se muestra la interfaz de HP WebInspect.

En HP WebInspect la configuración inicial requiere de poca aportación por parte del usuario y los informes que genera cuando ha terminado el análisis son muy completos [HP15].

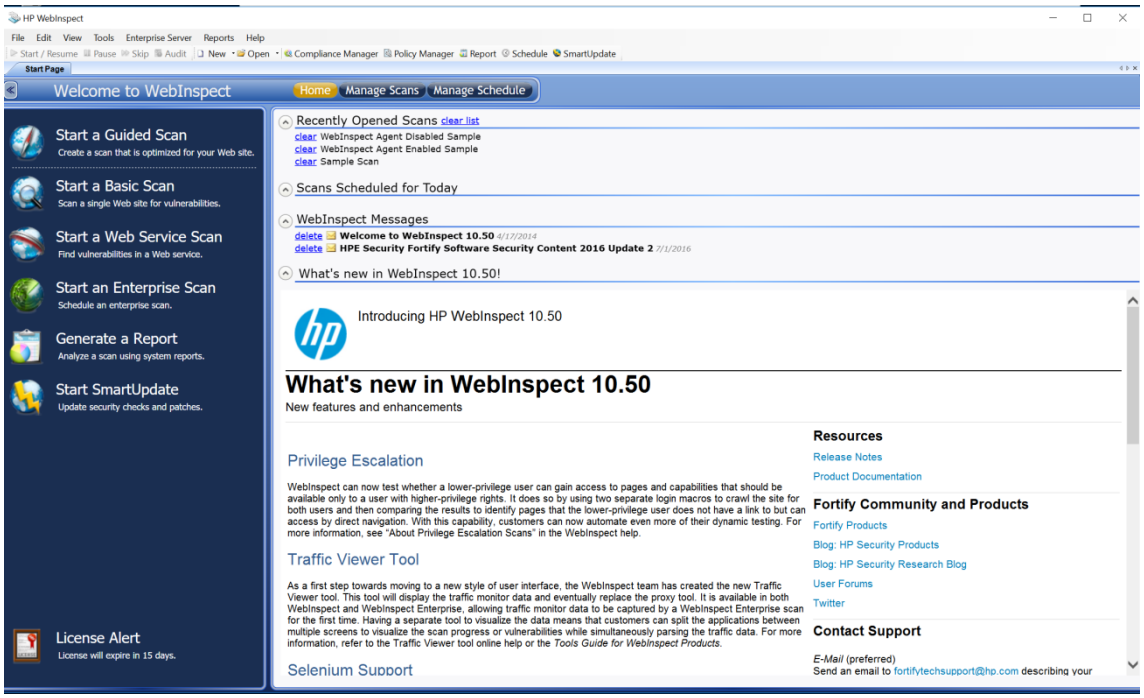


Figura 4.14: Captura de pantalla de la interfaz gráfica de HP WebInspect

HP WebInspect utiliza dos modos básicos para ejecutar su análisis:

- **Rastreo**: Es el proceso en donde HP WebInspect identifica la estructura que es objeto de análisis, en busca de enlaces, formularios, páginas y toda la estructura que compone a la aplicación.
- **Auditoría**: Este proceso realiza el rastreo de las vulnerabilidades en todas las páginas de la aplicación web recopiladas durante el rastreo.

HP WebInspect ofrece las siguientes características para realizar análisis de vulnerabilidades:

- **Informes**: Los informes en HP WebInspect pueden ser personalizados, se pueden generar informes orientados a determinadas audiencias indicando el nivel de información que se mostrará en cada caso. Los informes se presentan en diferentes formatos: *Portable Document Format* (PDF), HTML, Excel, Raw, *Rich Text Format* (RTF) o simplemente en formato texto. Además, se pueden incluir en los informes el resumen en gráficas del análisis de una aplicación.
- **Hacking Manual**: HP WebInspect contiene una herramienta que captura las peticiones enviadas y las respuestas recibidas de la aplicación analizada. A través de esto el usuario puede modificarlas y realizar ataques de forma manual.

- **Resumen y Reparación**: La información de vulnerabilidades que se muestran en la interfaz de HP WebInspect durante un análisis contiene las recomendaciones de reparación de cada una y también incluye instrucciones para evitar que se produzcan esos fallos a futuro. Toda esta información que se incluye es actualizada constantemente y de manera automática por parte de HP WebInspect.
- **Políticas de Rastreo**: En HP WebInspect se pueden modificar y adaptar las políticas de rastreo que tiene predefinida la herramienta. De esta forma se reduce el tiempo de análisis.
- **Detalles del Análisis**: La interfaz de usuario de HP WebInspect muestra durante todo el tiempo de análisis paneles que indican aspectos como:
  - **Sitio**: muestra la estructura jerárquica de los ficheros que componen la aplicación; también muestra por cada fichero las vulnerabilidades halladas en él y el código de estado HTTP de la respuesta del servidor.
  - **Secuencia**: muestra la información de los recursos del servidor hallados en la primera etapa de rastreo del análisis conducido por HP WebInspect.
- **Análisis de Servicios Web**: HP WebInspect también puede realizar búsquedas de vulnerabilidades en servicios web. Permite evaluar las solicitudes que contengan objetos de Servicios Web (*Simple Object Access Protocol* (SOAP)).
- **Exportación de Datos**: HP WebInspect cuenta con un asistente que permite exportar en formato XML toda la información recopilada durante un análisis, incluyendo comentarios, archivos ocultos, cookies, formularios, URLs, peticiones, sesiones. El usuario es el encargado de determinar la información que se exporta.
- **Configuración de Parámetros de Entrada**: Con la herramienta para el diseño de archivos de prueba HP WebInspect permite al usuario crear ficheros *.WSD que contienen los valores que HP WebInspect introducirá en los campos de los formularios que halle durante el análisis de una aplicación.
- **Agentes de Análisis Especializados**: HP WebInspect cuenta con agentes de búsqueda especialmente diseñados para realizar análisis sobre aplicaciones desarrolladas en entornos como IBM WebSphere, Adobe ColdFusion, Microsoft.NET, Lotus Domino, BEA Weblogic, Macromedia JRun, Oracle Application Server, Jakarta Tomcat.

4.5. Síntesis del Capítulo

El objetivo de este capítulo ha sido introducir el análisis de vulnerabilidades en aplicaciones web, haciendo hincapié en el análisis de caja blanca utilizando herramientas automáticas. Se ha comenzado con la exposición de las razones que justifican la existencia y estudio de este tipo de análisis. Seguidamente se han descrito los principales tipos de análisis y las características de las herramientas automáticas. Finalmente, se han descrito las funcionalidades de algunas de las herramientas más utilizadas por los profesionales de la seguridad de la información actualmente para realizar análisis de vulnerabilidades en aplicaciones web.

Capítulo 5

Trabajos Relacionados

Este capítulo describe el estado del arte de las principales técnicas de evaluación de las capacidades de detección de las herramientas de análisis de aplicaciones web. Empieza con la introducción de las principales guías elaboradas para la evaluación de estas herramientas. Seguidamente se exponen las técnicas existentes de evaluación y se indican los principales trabajos relacionados con cada una de ellas. El capítulo finaliza con una breve síntesis de lo expuesto en el mismo.

5.1. Guías de Evaluación

Existen determinados trabajos en los que se desarrollan diversas guías destinadas a comprobar las capacidades de las herramientas de análisis de vulnerabilidades en aplicaciones web. Uno de ellos destacado es el "Web Application Security Scanner Evaluation Criteria" (WASSEC) [Con09]. En esta guía se define una lista exhaustiva de características que deben tener estas herramientas. El objetivo es que cualquier usuario de las herramientas pueda evaluarlas. Las características se agrupan en ocho secciones, en las que cubre las dos fases pasiva y activa del análisis dinámico de vulnerabilidades. Estas secciones son:

1. Los protocolos que deben soportar: principalmente HTTP e *Hypertext Transfer Protocol Secure* (HTTPS).

2. Los mecanismos de autenticación que debe incluir son: básico, NTLM, por formulario, por certificado de cliente.

3. Los tipos de gestión de sesiones que debe permitir: cookie de sesión, por parámetros o por URLs.

4. Las opciones de rastreo que debe tener: especificar el host, indicar los valores de los parámetros, guardar la navegación del usuario.

5. Tipos de contenido que debe de poder analizar: HTML, XML, JavaScript.

6. Los tipos de vulnerabilidades que debe ser capaz de probar: XSS, SQLI, CSRF.

7. Las opciones de configuración que debe permitir. Por ejemplo, el programar los análisis o realizar varios análisis a la vez.

8. Los tipos de informe que debe realizar y su contenido.

En la Figura 5.1 se muestran estas secciones ordenadas temporalmente según el flujo de realización de un análisis de vulnerabilidades según WASSEC.

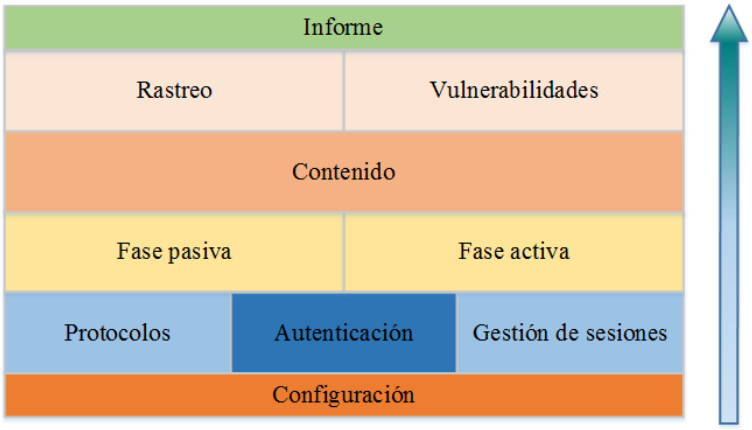


Figura 5.1: Secciones que deben tener las herramientas de análisis dinámico de vulnerabilidades según WASSEC

WASSEC fue desarrollada en 2009 y por el momento no ha sido actualizada, por lo que no incluye nuevas características que las herramientas actuales suelen incluir y que los profesionales de seguridad demandan de una herramienta de este tipo. Aunque esta guía es muy detallada y es de gran utilidad, hay muy poca información sobre los resultados de su aplicación en alguna herramienta.

Otra guía centrada en la evaluación de la efectividad de las herramientas de análisis de aplicaciones web es NISTSP. Al contrario que WASSEC, sólo define el conjunto mínimo de funcionalidades que deben incorporar las herramientas. Publicada en 2008, tampoco se ha actualizado desde entonces. Al menos hasta la fecha, no ha sido posible encontrar ningún artículo en el que se use para evaluar alguna herramienta.

Como parte del proyecto SAMATE el NIST desarrolló la guía [FO07] con las definiciones y funciones de estas herramientas de análisis. Incluye las definiciones de aplicación web, escáner o herramientas de análisis web e identifica una lista de vulnerabilidades que las herramientas deberían detectar. La lista que propone no está desarrollada, ya que sólo incluye vulnerabilidades habituales. Adicionalmente incluye referencias a otras clasificaciones como WASC o CWE.

Para probar las herramientas de seguridad y también como parte del proyecto SAMATE, el NIST ha creado la base de datos *Samate Reference Dataset* (SRD) que se describe en [BK05]. Es una recopilación de casos de prueba diseñados para comprobar las características de las herramientas de análisis, además de otras herramientas de seguridad. Inicialmente se desarrolló como un conjunto de casos de prueba en código

C, que posteriormente se fue ampliando a otros lenguajes de programación como PHP e incluso código binario. Incluye el código fuente de gran cantidad de fallos de seguridad diferentes, que se pueden usar para comprobar las herramientas de seguridad. En este caso tampoco se ha localizado ningún trabajo posterior sobre el uso de estos casos de prueba para comparar o evaluar herramientas de seguridad.

Aunque hay varias guías definidas, no hay ningún estándar que se use para valorar las capacidades de las herramientas. Por ello en [MB08] se utilizan los tipos de debilidades y fallos de seguridad definidos en el repositorio CVE, para desarrollar la lista CWE. El objetivo de CWE es obtener una lista de problemas de seguridad que pueda servir para medir las capacidades de las herramientas, de acuerdo con las necesidades de la industria. Aunque se actualiza frecuentemente, actualmente no se usa como un estándar de capacidad de las herramientas, al menos en lo que se refiere a las aplicaciones web. Las relaciones de CWE con otras clasificaciones de vulnerabilidades no se mantienen actualizadas. En este caso no sólo se incluye información para evaluar las herramientas dinámicas, sino también las estáticas.

5.2. Evaluación de Herramientas de Análisis Dinámico

En la última etapa del ciclo de desarrollo de las aplicaciones web es necesario incluir un análisis de vulnerabilidades. Solucionando los problemas que se encuentren, se reducirá la probabilidad de éxito de un potencial atacante y se reducirá el tiempo y esfuerzo en el desarrollo ya que permite enfocar el esfuerzo en otras tareas de seguridad más complejas o de otro tipo. Una de sus principales debilidades es la presencia de gran número de falsos positivos en sus resultados.

Hay varios trabajos relacionados con la evaluación de las capacidades de detección de vulnerabilidades de las herramientas de detección. Esta evaluación se puede hace principalmente por uno de los dos métodos siguientes:

1. Utilizando algún criterio o guía de evaluación ya definido.
2. Seleccionando un conjunto de herramientas de análisis y un conjunto de vulnerabilidades que deben detectar.

Una vez seleccionado el método se tienen dos opciones de revisión:

1. Revisar las herramientas manualmente para determinar si disponen de las capacidades necesarias para detectar las vulnerabilidades.
2. Seleccionar un conjunto de aplicaciones vulnerables en las que probar las herramientas y analizar el resultado.

En la Figura 5.2 se muestran las opciones disponibles.

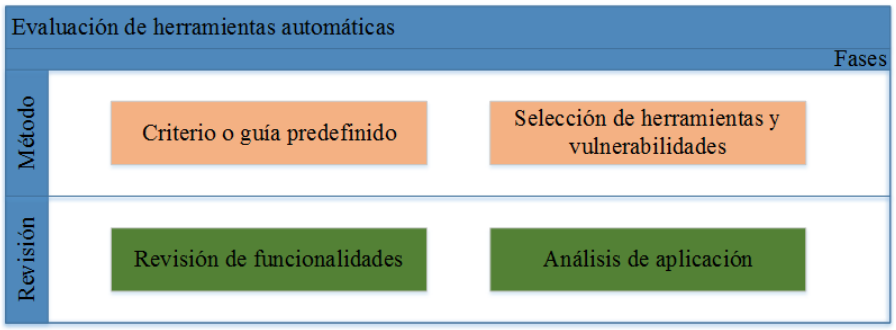


Figura 5.2: Opciones de evaluación presentes en herramientas de análisis dinámico

A continuación, se revisan los trabajos relacionados con la evaluación de herramientas automáticas para el análisis dinámico de aplicaciones, según las cuatro opciones indicadas:

- Siguiendo un criterio o guía previo y revisando las funcionalidades y características de las herramientas sin realizar el análisis de ninguna aplicación web, es decir, buscando entre las opciones que proporciona si sería capaz de encontrar cada tipo de vulnerabilidad que se haya tenido en cuenta.
- Siguiendo un criterio o guía previo y analizando aplicaciones web vulnerables.
- Sin un criterio previo, seleccionando un conjunto de herramientas a analizar y otro de vulnerabilidades, para a continuación comprobar si las herramientas incorporan funcionalidades o características que permitirían detectarlas, es decir, buscando entre las opciones que proporciona si sería capaz de encontrar cada tipo de vulnerabilidad que se haya tenido en cuenta.
- Sin un criterio previo, seleccionando un conjunto de herramientas a analizar y otro de vulnerabilidades implementadas en una aplicación vulnerable, que se analiza con las herramientas elegidas.

5.2.1. Evaluación Siguiendo un Criterio Previo

Existen varios artículos en los que se han empleado criterios de evaluación para determinar y comparar las capacidades de las herramientas automáticas. Los criterios que se han empleado principalmente han sido la guía WASSEC y la clasificación OWASPT10.

5.2.1.1. Revisión de Funcionalidades

En [Sae14a] se analizan herramientas de código libre siguiendo tres fases. Inicialmente se describe el criterio de evaluación a seguir, a continuación las herramientas a analizar y finaliza mostrando el resultado obtenido al aplicar el criterio sobre las herramientas. El criterio elegido es un subconjunto de WASSEC, que establece ocho secciones a tener

en cuenta en la valoración de las herramientas. En este trabajo se tienen en cuenta todas menos las dos últimas, esto es, las opciones de configuración y el informe final que hacen las herramientas no se evalúan. Para cada sección se especifican las características concretas que se van a buscar en cada una de las herramientas. Se buscan 11 protocolos, 7 mecanismos de autenticación, 6 sistemas de gestión de sesiones, 12 opciones de rastreo, 13 tipos de contenido y 33 tipos de vulnerabilidades. Con estas secciones del criterio WASSEC se evalúan 32 herramientas de código libre. Para cada una de ellas se indica la versión, el tipo de licencia y la fecha de la última actualización hasta la realización del artículo. En la fase de evaluación se revisan las herramientas buscando las características seleccionadas en cada sección. En este artículo no se evalúan las herramientas probándolas contra ninguna aplicación web, únicamente se revisan las opciones que proporcionan. Como resultado se valoran las herramientas según incluyan las características de cada una de las secciones, dando una puntuación a cada herramienta en cada sección. Se termina indicando la que obtiene mejor puntuación en cada sección y en total.

Un trabajo similar al anterior es [Sae14b]. Tanto las fases, como el criterio, secciones y características empleados son los mismos. La diferencia está en que, en lugar de evaluar herramientas de código abierto, en este otro artículo se evalúan herramientas comerciales. El criterio de evaluación es igualmente WASSEC, las secciones de este criterio son las mismas, es decir, todas excepto las dos últimas. Se evalúan las secciones de protocolos, mecanismos de autenticación, gestión de sesiones, rastreo, tipos de contenido y vulnerabilidades. No se evalúan la configuración y el informe. En cada sección las características buscadas son las mismas que en el trabajo anterior. En este caso la evaluación se conduce sobre 13 aplicaciones comerciales. El resultado es un valor numérico para cada herramienta y sección. Se concluye indicando la mejor herramienta en cada sección y la mejor herramienta de media en total.

En la Figura 5.3 se representa esquemáticamente las secciones de WASSEC que se han utilizado para evaluar las herramientas, tanto de código libre como comerciales, en estos dos trabajos en los que se han revisado manualmente sus características y funcionalidades para determinar si las herramientas serían capaces de detectar las vulnerabilidades que se han tenido en cuenta.

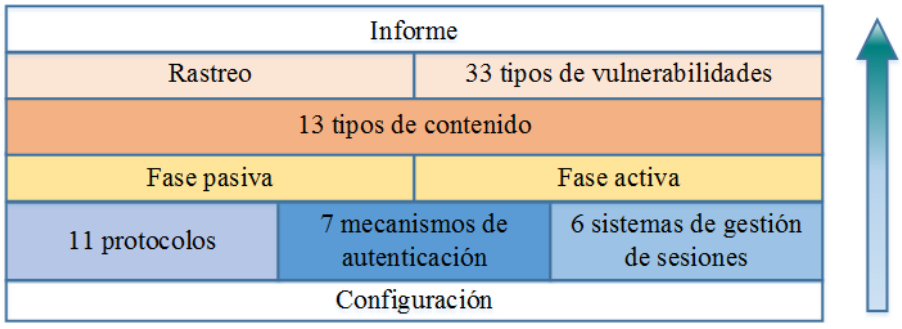


Figura 5.3: Secciones revisadas en [Sae14a] y [Sae14b]

5.2.1.2. Análisis de Aplicaciones Vulnerables

El criterio WASSEC se emplea en [GS11] para evaluar varias herramientas automáticas, pero únicamente la sección de vulnerabilidades. En este artículo adicionalmente se desarrolla una aplicación web que incluye diferentes tipos de vulnerabilidades. Este criterio de evaluación separa las vulnerabilidades en técnicas y lógicas:

- Las técnicas son las que se deben a errores técnicos como el filtrado incorrecto de la información que introduce el usuario. Dentro de las vulnerabilidades técnicas se pueden hallar vulnerabilidades como XSS o SQLI.

- Las lógicas son las que son resultado de fallos en los flujos lógicos de la aplicación. Dentro de las vulnerabilidades lógicas se considera, saltarse algún paso obligatorio en un proceso de varias fases.

En este trabajo se evalúan cuatro herramientas y de los resultados se concluye que las herramientas son bastante eficientes encontrando vulnerabilidades técnicas, pero no lógicas. Esto se debe a que incluyen bases de datos de vulnerabilidades conocidas, ampliables en mayor o menor medida, pero no es fácil que incluyan vulnerabilidades propias de la aplicación que se esté analizando.

La clasificación OWASPT10 se ha usado en varios artículos para evaluar las capacidades de herramientas de análisis de vulnerabilidades web, como es el caso de [DCV10]. En este artículo se emplea una aplicación vulnerable desarrollada expresamente para el trabajo que incluye las vulnerabilidades presentes en la clasificación OWASPT10 en su versión de 2010. Esta aplicación se ataca con las herramientas que se quieren evaluar. Se evalúan las dos fases de análisis dinámico de vulnerabilidades: la fase pasiva o rastreo y la fase activa o análisis. En esta segunda fase activa se analizan por separado las dos sub-fases que la componen, es decir, el envío de peticiones mal formadas y el análisis de las respuestas. Se analizan once herramientas, de las que se indica la versión analizada, el tipo de licencia y el precio en el momento del análisis. Cada una de las herramientas se ejecuta en tres modos diferentes, si lo permiten: inicial, con configuración y en modo proxy. En el primer modo se ejecutan las herramientas sin ningún tipo de configuración adicional, únicamente indicando la URL a analizar. En el segundo modo se facilitan a las herramientas las credenciales de acceso a la aplicación desarrollada. En el último modo se activan en las herramientas el modo proxy para que guarde las páginas que visita el usuario, así como sus credenciales. Al finalizar la navegación del usuario se pide a las herramientas que analicen la aplicación con esa información. El resultado son los tiempos de análisis empleados por cada herramienta en cada modo y las capacidades en cada fase. Para la fase pasiva se indican las páginas encontradas y accedidas en cada modo. Para la fase activa la tasa de vulnerabilidades no detectadas (falsos negativos) por cada herramienta, y las tasas de detección en cada uno de los modos de ejecución de cada una de ellas. Termina indicando los puntos de mejoras de las herramientas, que principalmente son dos: que no soportan el uso de código de cliente como JavaScript o Flash y que no son capaces de avanzar en la lógica de las aplicaciones sobre todo a las zonas que están detrás

de formularios complejos de varias fases y con un control fuerte del tipo de información de sus campos.

Otro artículo relevante en el que se utiliza la clasificación OWASPT10 para evaluar las capacidades de herramientas de análisis dinámico de vulnerabilidades es [FK11]. En este artículo se expone un trabajo de evaluación de once herramientas en la detección de cinco de las diez vulnerabilidades de la clasificación OWASPT10 de 2007, desarrollando una aplicación web vulnerable con esas cinco vulnerabilidades teniendo como requisitos adicionales que las vulnerabilidades se implementen de modo realista, como estaría en una aplicación de uso habitual, que incluya funcionalidades y tecnología de uso actual y que las vulnerabilidades estén claramente definidas y localizadas, para poder comprobar los resultados de las herramientas. Para cada una de las once herramientas se indica la versión analizada, el tipo de licencia y el precio. El resultado son las vulnerabilidades encontradas por cada una de ellas de las cinco implementadas para los modos de análisis elgidos en cada caso.

En la tesis [Mar12] se evalúan las capacidades de dos herramientas comerciales en la detección de las vulnerabilidades presentes en la versión de 2011 de la clasificación OWASPT10. En este caso se implementan las 10 vulnerabilidades de esta clasificación en una aplicación web. Los resultados incluyen, además de las detecciones de cada herramienta, los falsos positivos y los falsos negativos. Como conclusión se indican los dos principales puntos de mejora de estas herramientas. El primero de ellos se refiere a la fase de rastreo, en la que las herramientas no son capaces de llegar a todas las páginas de la aplicación. El segundo es la mala o inexistente interpretación , que hacen en algunos casos las herramientas, con lo que no se detectan vulnerabilidades que la herramienta sí prueba.

5.2.1.3. Esquema del Análisis con Aplicaciones Vulnerables

Como se ve en los trabajos relacionados indicados en este apartado, cuando se revisan herramientas de análisis utilizando aplicaciones web con vulnerabilidades, siempre se sigue el mismo esquema que puede verse en la Figura 5.4.

Los pasos que se siguen son:

1. Selección de un conjunto de herramientas a valorar.
2. Selección de un conjunto de vulnerabilidades que deberían detectar.
3. Desarrollo o selección de una o varias aplicaciones web que contienen esas vulnerabilidades.
4. Análisis de las aplicaciones web con las herramientas seleccionadas, con la configuración por defecto o con alguna configuración adicional.
5. Conclusiones: capacidades de detección de las herramientas y, en muchas ocasiones, sus posibles puntos de mejora.

Este esquema es válido tanto para las revisiones en las que se sigue un criterio, como en las que se selecciona un conjunto de vulnerabilidades, como se verá en un apartado

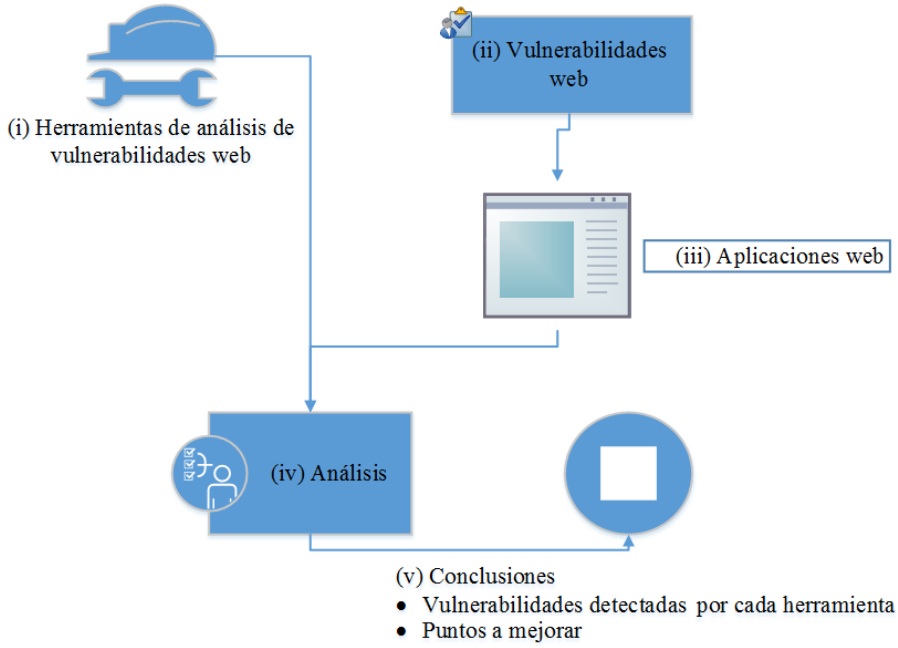


Figura 5.4: Evaluación usando aplicaciones vulnerables

posterior, siempre que se utilicen aplicaciones vulnerables para valorar las capacidades de las herramientas.

5.2.2. Evaluación Siguiendo un Criterio Propio

Existen adicionalmente muchos trabajos previos dedicados a evaluar herramientas de análisis de vulnerabilidades web siguiendo en la mayoría de los casos el método habitual de atacar con ellas aplicaciones vulnerables y, a continuación, valorar y comparar los resultados. Estos trabajos, a diferencia de los que tratan de seguir algún criterio de evaluación, se centran en alguna variable en concreto del análisis, ya sea la herramienta, las vulnerabilidades o la aplicación. Se pueden seleccionar las herramientas por el tipo de licencia o por sus características, las vulnerabilidades según sea su relevancia o difusión y las aplicaciones vulnerables dependiendo de si se han desarrollado vulnerables a propósito o tienen vulnerabilidades conocidas. En la Tabla 5.1 se lista y describe las opciones en cada caso.

Tabla 5.1: Opciones en la evaluación de herramientas

Variable	Opciones
Herramienta	Código libre Comerciales De Aplicaciones Web De Propósito General De Análisis de Tráfico
Vulnerabilidades	Todas las conocidas Las más relevantes Las que se encuentren en las aplicaciones vulnerables
Aplicación vulnerable	Vulnerable exprofeso de los fabricantes de herramientas Vulnerable exprofeso de terceros Conocida con vulnerabilidades

5.2.2.1. Revisión de Funcionalidades

Algunos trabajos analizan las capacidades de detección de varias herramientas sobre un amplio conjunto de vulnerabilidades. Este es el caso de [Che12] donde se analizan 64 herramientas frente a 33 vulnerabilidades. Las vulnerabilidades se analizan en forma de las capacidades de detección de las herramientas. Para la fase pasiva se utiliza una aplicación desarrollada para ese propósito, de forma que indica el porcentaje de páginas existentes a las que ha accedido la herramienta. Para la fase activa no se utiliza ninguna herramienta, lo que se hace es revisar las características de las herramientas. A diferencia de otros trabajos y artículos, en este caso la información se mantiene más actualizada ya que se revisa cada dos o tres años. Incluye la versión de los productos analizados, el tipo de licencia y su precio.

5.2.2.2. Análisis de Aplicaciones Vulnerables

En [Sut10] se analizan siete herramientas usando los propios sitios de pruebas de sus fabricantes, que contienen en total diez tipos distintos de vulnerabilidades. Los objetivos del trabajo son obtener el número de vulnerabilidades encontradas por las herramientas sin ninguna información adicional además de la URL inicial, el número de vulnerabilidades después de que la herramienta guarda la navegación de un usuario, si las vulnerabilidades encontradas son ciertas o no, y el tiempo empleado en comprobar la fiabilidad de la vulnerabilidad reportadas. En este trabajo se indican las ventajas e inconvenientes de cada una de las siete herramientas utilizadas y también el principal problema de estas herramientas, que se da en la ejecución de código cliente como JavaScript o Flash.

En [BBGM10] se hace uso de versiones de aplicaciones conocidas como Wordpress, Drupal o phpBB2, en las que ya existen vulnerabilidades conocidas. El objetivo es determinar y comparar las características de ocho herramientas en la detección de siete vulnerabilidades. Las principales carencias de las herramientas, según este trabajo, son la

ejecución del contenido activo, como el código JavaScript o el Flash, y la detección de las vulnerabilidades del tipo almacenadas. También se destaca que estas herramientas, utilizadas en conjunto sobre las aplicaciones web, sí proporcionan un alto grado de detección.

Algunos artículos sólo tienen en cuenta un conjunto muy reducido de vulnerabilidades. Por ejemplo, en [FVM07] y en [FVM14] sólo se evalúan las capacidades detectando XSS e SQLI. En [TWG13] se prueba una herramienta dedicada únicamente a buscar vulnerabilidades del tipo XSS. Y en [KZLR11] se comparan tres herramientas de análisis dinámico con el objetivo de determinar su fiabilidad detectando vulnerabilidades de los tres subtipos de SQLI: ciega, almacenada y reflejada.

Otros artículos están enfocados en entornos específicos. Por ejemplo, en [KSG+14] se analiza la interfaz web de una aplicación *Supervisory Control And Data Acquisition* (SCADA). En [DGdLO05] se clasifican las vulnerabilidades de los servicios web. En [ASW10] [ALY+15] se analizan las vulnerabilidades de aplicaciones sanitarias de código abierto. También hay artículos centrados en las herramientas de código libre o de bajo coste. Por ejemplo, [Mcq14] [Par15] recomiendan una combinación de estas herramientas y [NT10] proporciona una guía para seleccionarlas.

También hay trabajos en los que se evalúa un conjunto muy reducido de herramientas. Por ejemplo, en [MK15] se comparan únicamente dos herramientas de código libre analizando dos aplicaciones vulnerables a propósito y también revisando sus características. Y en [DBH14] se analizan tres herramientas, dos exclusivas de aplicaciones web y otra de propósito general, también en dos aplicaciones vulnerables, en este caso desarrolladas exprofeso para el artículo.

También hay trabajos relacionados con la detección de vulnerabilidades en aplicaciones web que utilizan algún sistema de detección de intrusión entre atacante y atacado, como *Network Intrusion Detection and Prevention System* (SNORT) [Roe16], para detectar los ataques. En [AIM14] se definen cinco reglas de SNORT para detectar ataques del tipo SQLI. Estas reglas se implementan en una instancia de SNORT que protege una aplicación vulnerable a propósito. Para atacar la aplicación lanzan manualmente un conjunto de 46 ejemplos ya existente de ataques de este tipo a diversas aplicaciones web. Como resultado se indica la tasa de detección y de falsos positivos. En [DAA13] se realiza un trabajo similar, pero ampliando las vulnerabilidades con XSS y ejecución de comandos, además de la SQLI. Como en el caso anterior, las pruebas consisten en atacar manualmente a la misma aplicación vulnerable a propósito, delante de la que se puesto SNORT, con las nuevas reglas.

5.2.3. Otros Trabajos de Evaluación

Además de los métodos anteriores de evaluación de herramientas de detección de vulnerabilidades web, existen otros trabajos en los que se propone el uso de algún otro método adicional a los utilizados.

Un caso particular de la evaluación de las herramientas de análisis es el uso de métricas borrosas [KLD10]. En este artículo se definen métricas borrosas para clasificar tanto las vulnerabilidades como las capacidades de detectarlas. Estas métricas borrosas se basan en

la dificultad de detectar la vulnerabilidad por las herramientas, asignándoles un valor a los cuatro parámetros: falso-positivo, falso-negativo, verdadero-positivo y verdadero-negativo. El objetivo es calificar cada par (vulnerabilidad, herramienta), aunque en el artículo se deja para un trabajo posterior. El método propuesto es potencialmente efectivo para evaluar las capacidades de las herramientas, pero implica mucho trabajo, al actualizarse continuamente las herramientas. Además, no deja claro qué aplicaciones vulnerables se deberían usar para evaluar herramientas y vulnerabilidades.

En [AKdLM10] se propone una guía para realizar un compendio de los requisitos de las herramientas de análisis. El objetivo es relacionar los requisitos de seguridad, definidos en la fase de análisis del ciclo de vida de desarrollo, con las pruebas definidas en la guía OWASPTGV3. Posteriormente habría que utilizar la herramienta que mejor se adaptara a esos requisitos para analizar la aplicación web. El principal beneficio de este método es que incluye el tratamiento de las vulnerabilidades en las etapas iniciales del desarrollo de la aplicación. La principal desventaja es que deja sin tratamiento las vulnerabilidades que no se hayan tenido en cuenta en la etapa de toma de requisitos.

5.3. Síntesis del Capítulo

El objetivo de este capítulo ha sido presentar los trabajos más representativos de la evaluación de herramientas de análisis dinámico de vulnerabilidades en aplicaciones web. Se ha comenzado con la exposición de las principales guías o estándares existentes para la evaluación de estas herramientas. A continuación, se han mostrado las distintas técnicas de evaluación, incluyendo una figura resumen de estas técnicas, para facilitar una visión general. Se ha continuado con la descripción de los principales trabajos relacionados, agrupados por la técnica de evaluación empleada.

Como se ha visto, existe gran cantidad de trabajos relacionados que evalúan las herramientas, aunque en la mayoría de ellos se sigue la misma técnica de evaluación. Esta técnica consiste básicamente en revisar los informes que proporcionan las herramientas sobre un conjunto de vulnerabilidades implementadas. En general no proporcionan información sobre cómo funcionan las herramientas ni proponen mejoras para solucionar las carencias detectadas.

Capítulo 6

Mejora de las Herramientas de Análisis

Este capítulo presenta las mejoras desarrolladas para dar respuesta y solucionar las principales debilidades detectadas en las herramientas de análisis dinámico de aplicaciones web. El capítulo comienza indicando estas debilidades en las dos fases de análisis dinámico, la fase pasiva y la fase activa. Posteriormente, se describen las soluciones desarrolladas para cada una de ellas y los algoritmos con los que se implementan. El capítulo finaliza con un breve resumen de lo expuesto en el mismo.

6.1. Puntos de Mejora de las Herramientas de Análisis

Las principales debilidades de las herramientas de análisis dinámico de vulnerabilidades se agrupan según la fase del análisis en la que afecte. En el caso de la fase pasiva o fase de rastreo, las principales debilidades están relacionadas con la dificultad que tienen estas herramientas para comportarse como un usuario humano. Esta dificultad es doble; por un lado, hay una dificultad técnica y por otro de conocimiento. La dificultad técnica tiene que ver con que las características de navegación automática de estas herramientas no incorporan todas las funcionalidades de la misma forma en que las incorporan los navegadores web de uso habitual. Esto es especialmente relevante cuando tienen que ejecutar código de cliente, sobre todo en el caso de código JavaScript, ampliamente utilizado. Este código de cliente se encarga de modificar la página visitada según las acciones del usuario, sin necesidad de la intervención del servidor o mediante respuestas XML si se utiliza AJAX. Las herramientas no son capaces de tratar correctamente estas acciones al no generar peticiones similares a las de un usuario humano.

La dificultad de conocimiento tiene relación con la lógica de la aplicación web. Las herramientas de análisis desconocen la lógica que hay dentro de las aplicaciones web y no actúan en consecuencia a ella, como haría una persona. En particular, no son capaces de realizar las acciones de las aplicaciones que impliquen un flujo de acciones en varios pasos y en las que se tenga que rellenar formularios con fuertes restricciones en sus campos. Por ejemplo, una herramienta de este tipo no será capaz de dar de alta un usuario en una aplicación, especialmente si el usuario tiene que introducir valores en varios campos en los

que se revisa que sean del tipo esperado; por ejemplo; que en el campo para el teléfono se ponga un número de teléfono, que en el campo del número de tarjeta de crédito un número que pueda ser de una tarjeta, o que se ponga una población que pertenezca a una provincia seleccionada con anterioridad.

En cualquiera de los dos casos anteriores, la ejecución incorrecta del código de cliente y en no poder avanzar por las acciones definidas en la aplicación, la consecuencia es que la herramienta automática no será capaz de llegar a todas las páginas que componen la aplicación. Si no llega a todas las páginas de la aplicación, no será capaz de encontrar todos los puntos de entrada a la misma, en los que probar vulnerabilidades en la siguiente fase activa. De esta forma muchas vulnerabilidades potenciales se quedarán sin probar y posiblemente sin detectar.

Por último, en el caso de la fase activa o de prueba y análisis, el principal problema no está relacionado con la forma en la que estas herramientas prueban cada una de las diferentes vulnerabilidades que puedan tener las aplicaciones web, sino con saber qué vulnerabilidades deberían de detectar, o al menos probar, las herramientas de análisis, y si realmente las llegan a probar. Hasta el momento no existe ninguna lista actualizada y completa que indique los tipos de vulnerabilidades que las herramientas deberían ser capaces de detectar, ni por lo tanto una aplicación o conjunto de aplicaciones vulnerables que las contengan. Además, tampoco hay información sobre las pruebas que realmente hacen las herramientas durante esta fase, únicamente se tienen los informes resultantes de atacar aplicaciones vulnerables, con las vulnerabilidades detectadas, sin incluir las que se han probado.

En este caso, la consecuencia de no tener una lista previa de tipos de vulnerabilidades que las herramientas deberían probar y de no saber si realmente las prueban o no, es que no existe un criterio definido para analizar las capacidades de las herramientas y compararlas entre ellas. Con la existencia de ese criterio será posible comparar los resultados de unos trabajos con otros. Por otro lado, sabiendo qué pruebas realizan realmente las herramientas, se detectarán puntos de mejorar en ellas, además de poder saber qué herramienta es más conveniente en cada caso. A un analista de seguridad que use estas herramientas le será de gran ayuda saber qué herramientas es mejor probando y detectando los tipos de vulnerabilidades que le interesen.

6.2. Mejoras para la Fase Pasiva

Lo que buscamos en este apartado es una solución que mejore las dos debilidades de las herramientas en la fase pasiva del análisis automático descritas en el apartado anterior: ejecución del código de cliente como lo haría un navegador habitual y navegar avanzando por flujos definidos en la aplicación. En concreto la solución tiene que ejecutar el código JavaScript de las páginas visitadas y encontrar valores válidos para los campos de los formularios web que haya en esas páginas, de forma que la herramienta pueda seguir navegando por la aplicación como lo haría un usuario humano. Estos valores deberán cumplir los siguientes requisitos:

1. Ser del tipo del campo tanto en formato como tamaño; por ejemplo, si un campo se

llama “email”, habrá que poner algo que parezca una dirección de correo electrónico.

2. Se deberán tener en cuenta las relaciones entre campos; por ejemplo, el valor de un campo “localidad” deberá ser consecuente con el valor de un campo anterior “provincia”.

3. Será necesario tener un criterio para determinar si los valores enviados son correctos.

En la aportación que se describe a continuación se han tenido en cuenta las siguientes consideraciones:

1. Se van a rellenar los campos en el mismo orden que lo haría un usuario humano, de arriba abajo.

2. No se van a modificar los valores de los campos ocultos.

3. No se intentarán resolver los campos de *Completely Automated Public Turing test to tell Computers and Humans Apart* (CAPTCHA).

4. No se van a atacar los formularios de inicio de sesión donde hay que introducir las credenciales.

Para cada uno de los requisitos indicados en el apartado anterior, se definen las siguientes características de la solución propuesta:

I. Los valores de los campos de texto se extraen de una fuente de información externa. En este caso se usarán las Tablas de Fusión de Google [GHJ+10], donde para cada campo de texto se intentará extraer un valor de ese tipo, que al ser del tipo buscado en general cumplirá con los requisitos. Siguiendo con el ejemplo del apartado anterior, al buscar una tabla de fusión con un campo “email” seguramente los valores que se encuentren en ese campo de la tabla sean realmente direcciones de correo electrónico.

II. Para obtener campos de texto relacionados entre ellos, por ejemplo, una población con su provincia, también sirve la información que proporcionan las Tablas de Fusión, ya que si se buscan tablas con los campos “provincia” y “población” seguramente los registros de la tabla que se encuentren cumplan la relación.

III. Si la relación se da en campos de selección es habitual que, por ejemplo, al seleccionar una provincia, la página se descargue mediante AJAX sus poblaciones para seleccionar una de ellas; por ello estos valores se obtienen simulando un navegador que vaya seleccionando valores de los campos de selección que los tengan, hasta que se haya obtenido un valor para cada uno.

IV. Para determinar si el envío de los valores ha tenido éxito se considera el criterio siguiente: si los valores son válidos y suficiente, llevarán a un nuevo contenido y serán los valores buscados; si son válidos pero insuficiente (cuando falta alguno), devolverá la misma página con los campos correctos con valor ya rellenos; y si no son válidos, devolverá el mismo formulario, con los campos inválidos vacíos. Por

simplicidad se han tratado sólo estas tres opciones, aunque en cualquiera de los dos últimos casos, también la aplicación puede devolver una página de error.

V. En el caso particular de los campos de contraseñas se ha optado por generar una contraseña de cierta complejidad para cada formulario que tenga un campo de este tipo, colocando el mismo valor en todos los campos de contraseña que tenga ese formulario.

A partir de estas características tenemos la descripción del algoritmo para el preproceso de formularios web (véase Figura 6.1):

a. Inicialmente se analiza la página, registrando los formularios y sus campos de los que se guardan características como: oculto o no, tipo, valores por defecto, valores de todos los campos de selección (ejecutando el código AJAX si es necesario, como se indica en III, o el literal asociado (ver punto siguiente).

b. Para buscar los campos en las Tablas de Fusión se comienza buscando el nombre del campo, pero como en muchos casos el nombre asignado en el formulario no se corresponde con el literal que se muestra, como segunda opción se busca el literal más parecido usando la distancia de Levenshtein [LL07]; en tercer lugar, el literal más cercano como en [HHLT03] y, por último, se buscan sinónimos en [Lab12].

c. Cuando se localice una Tabla de Fusión que tenga como columna el campo buscado, o (b), se toman n valores para él (por ejemplo, n = 5), y también para todos los demás campos de texto del formulario que aparezcan en esa tabla.

d. Comenzando con el primer campo se van buscando valores correctos para cada uno de ellos. En cada iteración se envían los campos con valores ya encontrados como correctos según IV, junto con el siguiente campo sin valor válido de la siguiente forma: si es un campo de selección se toma un valor de (a); y si es un campo de texto se envía ese campo, junto con los demás campos de texto todavía sin valor encontrado, con los valores que se hayan obtenido en (c).

e. Si los valores de (c) no son correctos, en la siguiente iteración volverá a estar ese campo sin valor correcto, y se usarán valores distintos a los encontrados en la iteración anterior.

f. Si el formulario es de alta, por ejemplo, en el registro de usuarios, al encontrar los valores correctos para cada campo, seguramente se realice el alta del usuario, por lo que no podrán usarse una segunda vez. En este caso se propone guardar el siguiente valor de la Tabla de Fusión para cada valor válido.

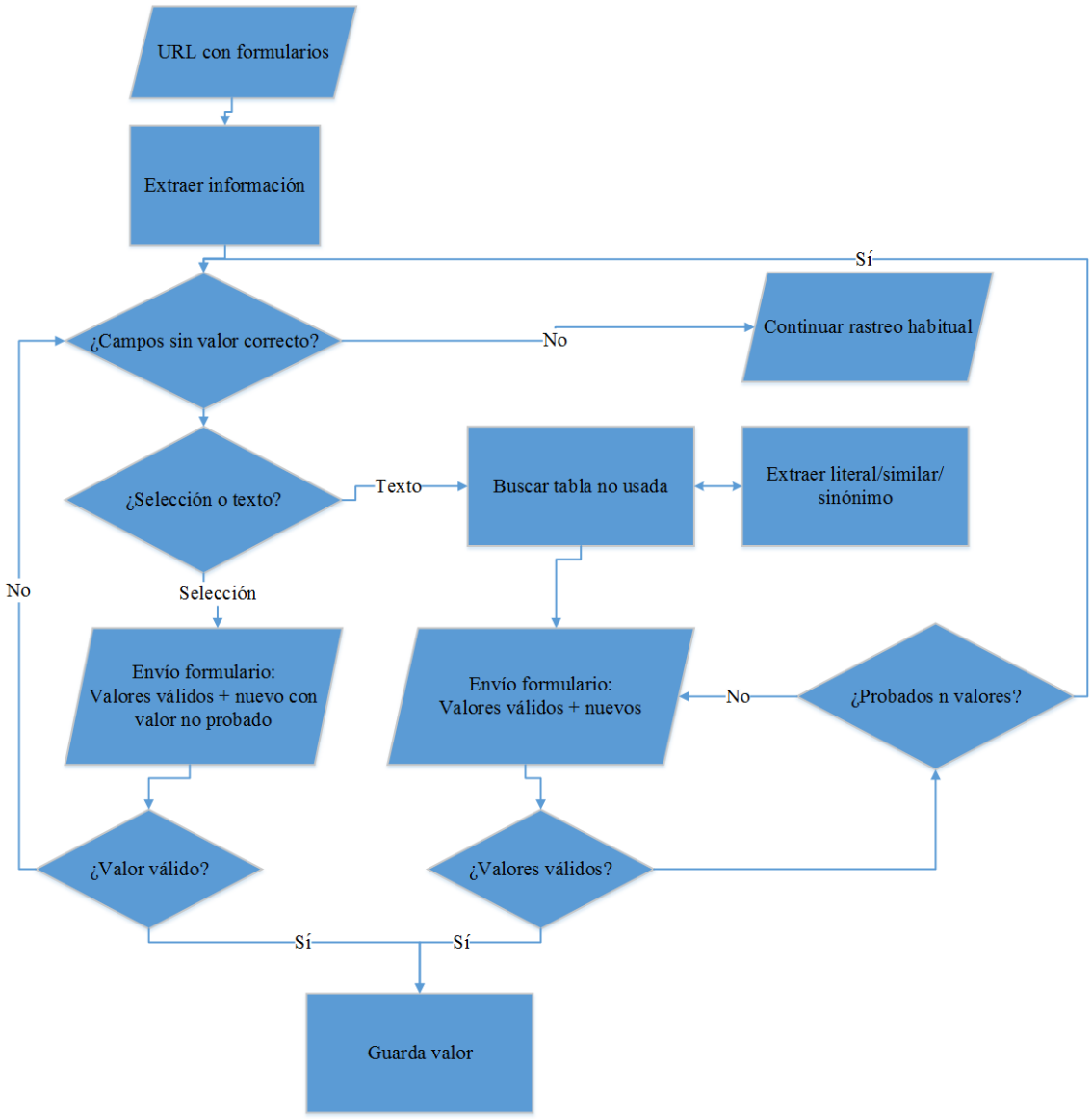


Figura 6.1: Preproceso de formularios web

6.2.1. Tablas de Fusión de Google

Esta funcionalidad de Google permite a los usuarios subir archivos en formato de tabla; por ejemplo, hojas de cálculo o archivos *Comma-Separated Values* (CSV). Estas tablas pueden contener, además de números y texto, objetos como líneas y polígonos. El sistema permite la integración de información a partir de diversas fuentes y proporciona varias formas para la visualización, el filtrado y la consulta de la información. En este último caso, el de consulta, se puede hacer directamente a través de una página web o mediante algún programa que haga uso de la API que proporciona. Aunque sobre estas tablas se puede implementar un mecanismo de control de acceso, existe una gran cantidad de ellas de uso libre. Actualmente existen muchas de ellas que contienen información sobre muchos dominios de conocimiento. En este caso, mediante el uso de la API, se realizan consultas buscando los nombres de los campos de los formularios, para recuperar valores que sean válidos en esos campos y que la aplicación acepte como buenos.

6.2.2. Distancia de Levenshtein

La distancia de Levenshtein, también conocida como distancia de edición o distancia entre palabras, es el número mínimo de operaciones requeridas para transformar una cadena de caracteres en otra. Se entiende por operación la inserción, eliminación o sustitución de un carácter. Utilizando esta distancia, dos cadenas de texto serán más similares cuanto menor sea el número de operaciones de edición necesarias para transformar una en otra, o viceversa.

En este caso se usa la distancia de Levenshtein para buscar, como segunda opción, los nombres de los campos de los formularios en las Tablas de Fusión de Google. La primera opción es buscar directamente el nombre del campo, pero en muchos casos este nombre no se corresponde con el literal que ve el usuario, aunque sí se parece, siendo una contracción o una palabra similar. Por ello, dentro del código de la página web se buscará el literal más cercano, según esta distancia, al nombre del campo para el que se quiere obtener valores. Por ejemplo, si un campo de formulario se llama “nombreusuario” esta búsqueda podrá localizar el literal que aparece realmente en la página si es por ejemplo “Nombre” o “Usuario”.

6.2.3. Literal más Cercano

Para localizar el literal más cercano en una página web se utiliza un analizador del DOM. Se envía el código HTML de la página al analizador del DOM, el cual calcula la localización de cada elemento en la pantalla y la distancia a todos los demás elementos. A continuación, se localiza el texto que contiene el elemento más cercano a cada campo de formulario, para considerar su descriptivo y buscarlo en las Tablas de Fusión de Google, como tercera opción de búsqueda de valores.

6.3. Mejoras Propuestas para la Fase de Ataque y Análisis

Esta sección describe las propuestas desarrolladas para mejorar la segunda fase del análisis de vulnerabilidades de aplicaciones web. Estas propuestas son, en primer lugar, el método desarrollado para conseguir tener una clasificación actualizada de vulnerabilidades web, y, en segundo lugar, un mecanismo para poder determinar qué pruebas realizan realmente las herramientas automáticas. La clasificación actualizada y unificada de vulnerabilidades servirá de criterio para poder evaluar las capacidades de detección de las herramientas de análisis y para saber qué vulnerabilidades debe tener una aplicación vulnerable. Con el mecanismo de pruebas se podrán analizar las herramientas para conocer su funcionamiento y detectar sus carencias.

6.3.1. Clasificación de Vulnerabilidades

Esta sección describe el método desarrollado para conseguir tener una clasificación actualizada de vulnerabilidades web y un mapeo de las clasificaciones existentes. El punto clave de este método es la simplificación de cada vulnerabilidad distinta a un conjunto reducido de palabras que la describan de forma única y que pueda usarse a continuación para buscar en Internet relaciones entre las clasificaciones.

(1) El primer paso para obtener una clasificación de vulnerabilidades actualizada consiste en seleccionar las últimas versiones de las clasificaciones de vulnerabilidades en aplicaciones web. Esta tarea se lleva a cabo buscando clasificaciones de vulnerabilidades en las organizaciones más conocidas como OWASP, WASC o MITRE, ya mencionadas anteriormente.

(2) El objetivo del segundo paso es encontrar mapeos entre clasificaciones proporcionados por las propias organizaciones que elaboran las clasificaciones. Es habitual que adicionalmente al desarrollo de una clasificación de vulnerabilidades, estas organizaciones intenten relacionar sus vulnerabilidades con las que aparecen en otras clasificaciones. Esta relación puede estar en un único documento que incluye las relaciones, o como parte de un conjunto de documentos.

(3) El tercer paso consiste en encontrar relaciones entre las vulnerabilidades de diferentes clasificaciones en otras organizaciones o estudios previos. Existen organizaciones y estudios previos que no tratan de elaborar su propia clasificación, sino que proporcionan información sobre las relaciones entre las vulnerabilidades de diferentes clasificaciones. A partir de estas relaciones se elabora una lista de vulnerabilidades en las que se agrupan bajo una misma entrada las que están relacionadas.

(4) El último paso trata de buscar relaciones adicionales entre vulnerabilidades de diferentes clasificaciones. Estas relaciones adicionales se pueden encontrar en los sitios web de las organizaciones que elaboran las listas o en otras páginas de Internet. Estas páginas, de las organizaciones u otros sitios web, son páginas que tratan una vulnerabilidad en exclusiva e intentan aportar toda la información disponible sobre ella. Para encontrar estos últimos mapeos en este trabajo se propone reducir

las descripciones que existan de cada vulnerabilidad a un conjunto reducido de palabras que la describan de forma única, con el objetivo de que no existan dos vulnerabilidades con las mismas palabras asignadas. A continuación, estas palabras se usan para buscar en Internet mapeos con las vulnerabilidades de las clasificaciones seleccionadas.

En el paso final de la versión del algoritmo que se describe, se buscan relaciones en las páginas de las organizaciones que hacen las clasificaciones, relaciones adicionales a las que proporcionan en forma de mapeos entre clasificaciones, y que suelen estar incluidas en información detallada que elaboran sobre una vulnerabilidad concreta.

En la Figura 6.2 se explican esquemáticamente estos pasos.

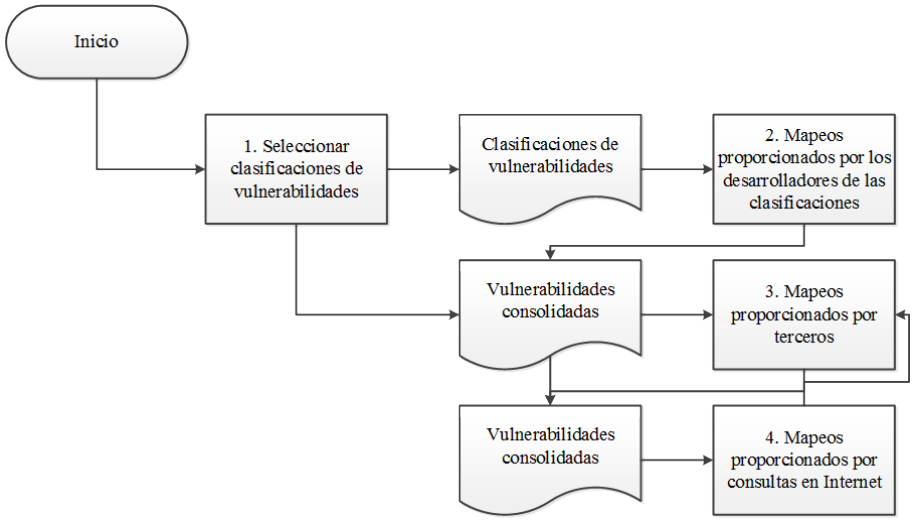


Figura 6.2: Algoritmo para unificar las clasificaciones actuales

En la Figura 6.3 las tareas que se realizan en el último paso (4).

El objetivo del paso (4) del algoritmo descrito anteriormente es localizar relaciones que puedan existir entre vulnerabilidades, pero que no se encuentran en los repositorios conocidos de relaciones. Son relaciones que se encuentran habitualmente en páginas web que tratan en detalle una única vulnerabilidad y que incluyen información sobre las vulnerabilidades de otras clasificaciones existentes con las que estén relacionadas.

Para buscar este tipo de relaciones entre vulnerabilidades se buscan en Internet los prefijos de los códigos de las vulnerabilidades de cada clasificación, por ejemplo “OWASP-‘’ o “WASC-‘’, junto con la descripción de cada vulnerabilidad. Las respuestas obtenidas se analizan para encontrar los mapeos.

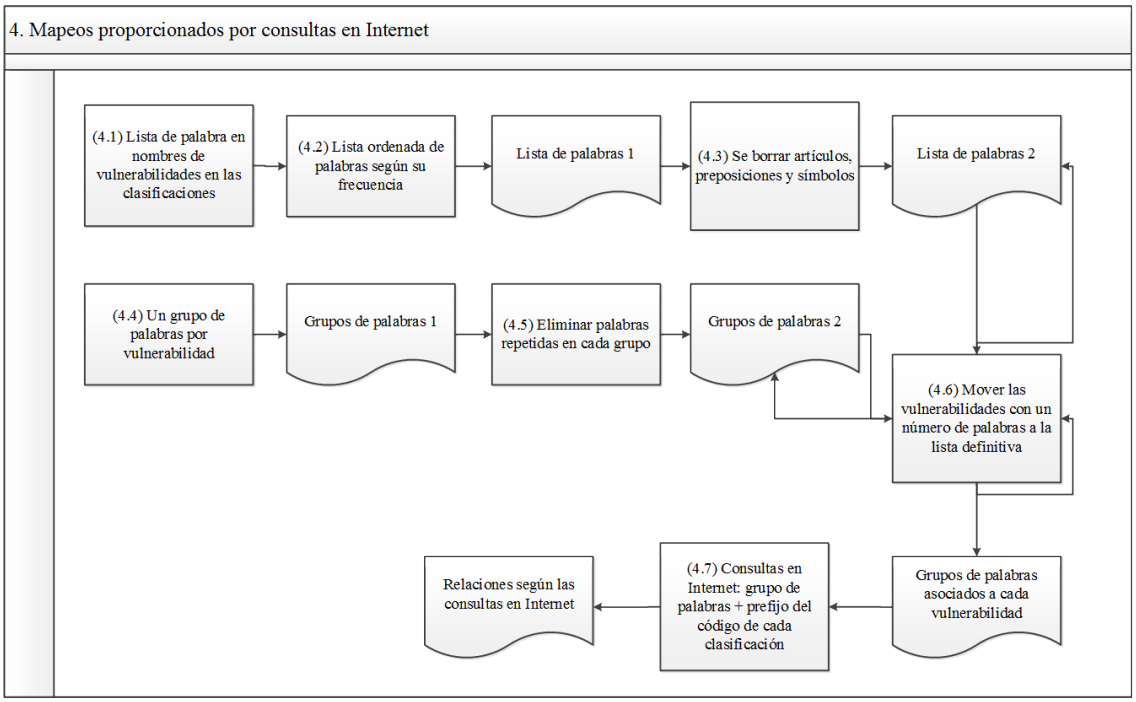


Figura 6.3: Búsqueda de relaciones entre vulnerabilidades en Internet

El problema en este punto es determinar qué se llama la descripción de una vulnerabilidad. En las clasificaciones cada código de vulnerabilidad, por ejemplo "OTG-INPVAL-006" en guía de pruebas OWASPTGv4 o "WASC-19" en WASCTC, lleva asociado una frase o un conjunto de palabras que la describen, habitualmente todas en el idioma inglés. Estas descripciones pueden contener palabras que se repitan en varias clasificaciones para una misma vulnerabilidad, como por ejemplo "SQL injection", pero también en otros casos cada clasificación asocia una descripción con palabras distintas para la misma vulnerabilidad; por ejemplo, OWASP relaciona la vulnerabilidad "Testing for CAPTCHA" de su guía OWASPTGv3 con "Testing for Weak password policy" de OWASPTGv4.

Para resolver este problema se propone reducir las descripciones existentes de cada vulnerabilidad a un conjunto mínimo de palabras que la describa de forma única. Estas palabras están asociadas a la vulnerabilidad y deberían aparecer si se busca esa vulnerabilidad y no aparecer al buscar otra vulnerabilidad distinta.

Para encontrar estas palabras clave de cada vulnerabilidad se aplica la idea descrita en [JLM14], donde se considera que las palabras que se repiten con más frecuencia son las menos importantes al ser las que menos ayudan a discriminar un concepto de otro. Aplicando esa idea a las descripciones de vulnerabilidades, se tiene que las palabras clave de una vulnerabilidad serían las menos frecuentes y las que podrían usarse para discriminar una vulnerabilidad de otra. El conjunto mínimo de palabras de una vulnerabilidad se obtendrá a partir de todas las palabras incluidas en todas sus descripciones en el mismo idioma (siempre estarán en inglés).

Para encontrar el conjunto mínimo de palabras asociado a cada vulnerabilidad se siguen los siguientes pasos:

(4.1) Elaborar una lista que incluya todas las palabras que aparecen en cada una de las descripciones de todas las vulnerabilidades en las clasificaciones seleccionadas en el apartado (1).

(4.2) Calcular la frecuencia de aparición de cada palabra en la lista obtenida en el paso anterior (4.1) y elaborar una segunda lista en las que aparezca cada palabra y su frecuencia, ordenadas por este último valor.

(4.3) Eliminar artículos, preposiciones y caracteres como comas, corchetes y comillas.

(4.4) Para cada vulnerabilidad distinta obtenida en el paso (3) se obtiene un conjunto de palabras formado por todas las descripciones que aparezcan en alguna de las clasificaciones donde figure.

(4.5) Se eliminan las palabras repetidas del conjunto asociado a cada vulnerabilidad.

(4.6) A partir del conjunto de palabras de las descripciones de cada vulnerabilidad hay que conseguir un conjunto distinto de palabras para cada una de ellas. Para ello primero los conjuntos que tienen el número buscado de palabras se llevan a la lista definitiva. A continuación, la palabra con mayor frecuencia se elimina de todos los grupos restantes y también de la lista de palabras. Este paso se repite hasta que todos los grupos de palabras asociados a vulnerabilidades tienen el número de palabras buscado. De esta forma con cada iteración del punto (4.6) se van dejando las palabras menos frecuentes, que son las que mejor describirán las vulnerabilidades de forma única. La lista de grupos de palabras que resumen cada vulnerabilidad será la lista de vulnerabilidades definitiva.

7. Por último, a partir del conjunto reducido de palabras que describe cada vulnerabilidad, se tienen que encontrar nuevos mapeos de las vulnerabilidades con las de las clasificaciones seleccionadas. Para ello se realizan consultas en Internet buscando en los sitios web de las clasificaciones los prefijos de los códigos de las vulnerabilidades junto con la descripción resumida.

6.3.2. Aplicación Vulnerable

Una vez que se tiene una lista unificada con las vulnerabilidades que debería ser capaz de probar y detectar una herramienta de análisis de vulnerabilidades, el siguiente paso será disponer de una aplicación vulnerable que contenga todas las que sea posible de esa lista. De esta forma, se podrán probar las herramientas y comparar bajo un mismo y más amplio criterio.

Para obtener esta aplicación, o conjunto de aplicaciones vulnerables, el primer paso es revisar las aplicaciones web vulnerables que existen actualmente. Para esto se ha hecho una recopilación de proyectos de distintas organizaciones y sitios o trabajos de recopilación

de aplicaciones vulnerables conocidos en el ámbito de la seguridad web. Estas aplicaciones tienen como objetivo probar herramientas y apoyar en la enseñanza en seguridad web.

Las aplicaciones se han revisado buscando las siguientes características:

1. Deben tener claramente definidas las vulnerabilidades.
2. Deben ser fácilmente modificables para poder agregar nuevas vulnerabilidades.
3. Deben representar a las aplicaciones actuales en términos de funcionalidad y tecnología.
4. Deben contar con vulnerabilidades de distintos tipos.
5. Deben contar con buena documentación.

Como aplicaciones vulnerables relevantes tenemos las que se indican a continuación, descritas anteriormente:

- **WebGoat** [OWA16b].
- **Mutillidae II** [OWA16a].
- **Damn Vulnerable Web Application** [Ran16].
- **The ButterFly Security Project** [Sof16].
- **WackoPicko** [Dou16].

Las aplicaciones que se han elegido cumplen con lo especificado. Algunas de estas aplicaciones han sido desarrolladas como trabajos de investigación y otras por empresas que se dedican al análisis de vulnerabilidades. Las aplicaciones pueden ser fácilmente instaladas en distintas plataformas. En la Tabla 6.1 se indica el número de vulnerabilidades que contiene la versión analizada de cada una de ellas.

Tabla 6.1: Cantidad de vulnerabilidades en las aplicaciones seleccionadas

Aplicación	**Vulnerabilidades**
WebGoat 5.4	40
Mutillidae 2.6.8	38
Damn Vulnerable Web Application 1.0	17
The ButterFly Security Project 1.0	36
WackoPicko 1.0	11

No se han considerado aplicaciones a las que se le ha encontrado alguna vulnerabilidad como Joomla o Wordpress. En estas aplicaciones no se tiene la certeza de si han sido detectadas todas las vulnerabilidades que puedan existir en la aplicación.

Aplicaciones que no tienen una buena documentación, que cuentan con pocas vulnerabilidades, que no representen aplicaciones con tecnologías actuales o que no hayan sido actualizadas en mucho tiempo tampoco han sido consideradas.

Las aplicaciones con mayor número de vulnerabilidades de la lista son WebGoat y Mutillidae II. Entre las dos tienen un total de 46 vulnerabilidades diferentes. Butterfly podría ser tenida en cuenta, pero debido a que ya se tiene una aplicación desarrollada con el lenguaje PHP no ha sido considerada al tener menos vulnerabilidades que Mutillidae II.

En la Figura 6.4 se observa el proceso de selección y los filtros por los que tuvieron que pasar las aplicaciones web vulnerables a propósito. A pesar de que existen muchas aplicaciones desarrolladas por instituciones, organizaciones, universidades y algunas en trabajos de investigación, éstas no cuentan con lo necesario para pasar los filtros definidos.

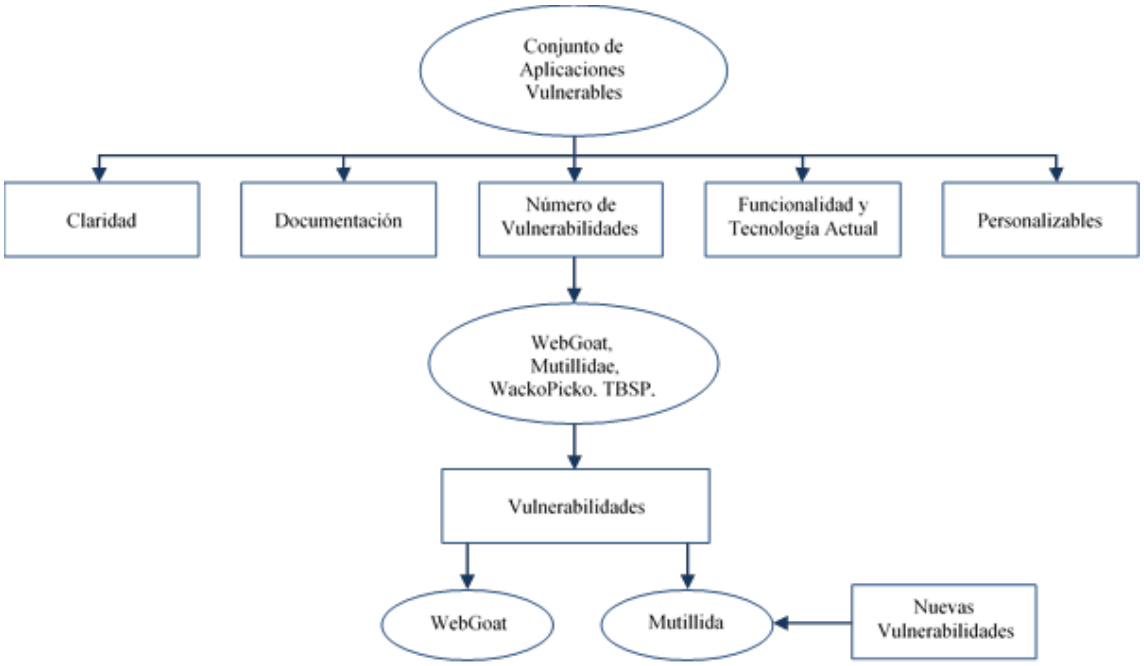


Figura 6.4: Selección de aplicaciones vulnerables a propósito

6.3.3. Evaluación de las Herramientas

La metodología utilizada en la literatura para la evaluación de las herramientas se apoya únicamente en los resultados del informe obtenido por la herramienta de análisis de vulnerabilidades que se esté evaluando al analizar una aplicación web. En la Figura 6.5 se presentan los componentes que conforman el modelo tradicional.

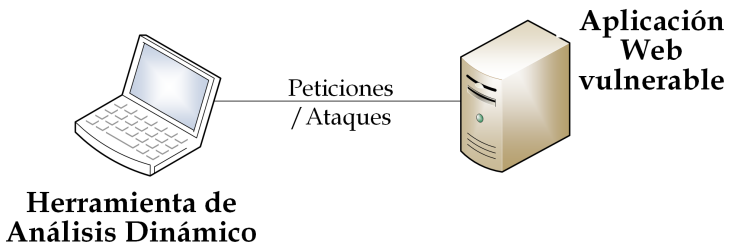


Figura 6.5: Modelo tradicional de evaluación

El informe que proporcionan las herramientas se revisa para verificar las detecciones reportadas. Posteriormente, las detecciones de cada una de ellas se comparan con las del resto de herramientas evaluadas. En la Figura 6.6 se muestra el diagrama del flujo que se sigue en este tipo de evaluaciones.

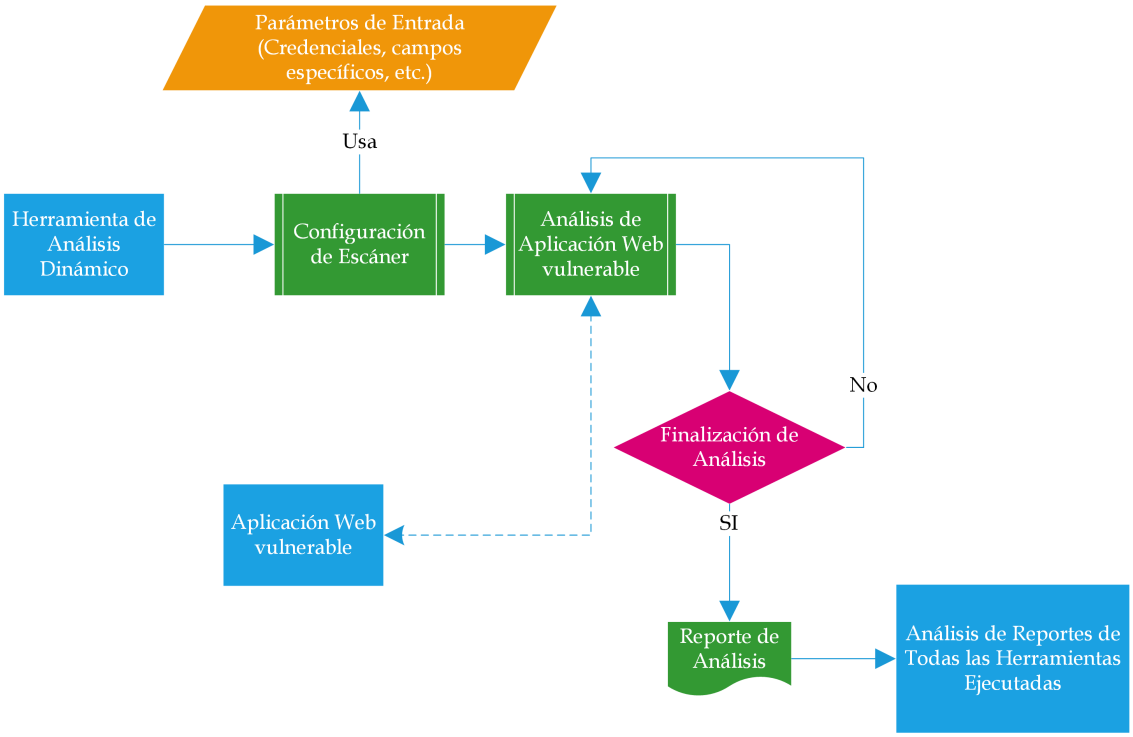


Figura 6.6: Diagrama del modelo tradicional

Para obtener un panorama más claro respecto al funcionamiento y precisión de estas herramientas, se aplica un enfoque distinto al que en trabajos previos se ha realizado, introduciendo en la estructura típica de estudio, un sistema de detección de intrusos IDS, que permita obtener las solicitudes de ataque que cada herramienta realiza en su análisis y con esto contrastar junto al informe correspondiente. De esta manera se podrá obtener información que determine la eficiencia de las herramientas para detectar vulnerabilidades.

En la Figura 6.7 se muestra la estructura de la propuesta.

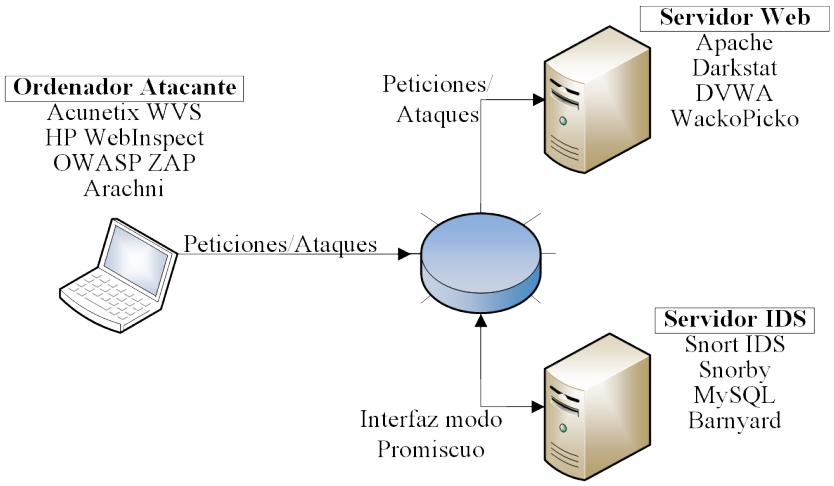


Figura 6.7: Modelo propuesto de evaluación

El modelo propuesto añade como componente un sistema de detección de intrusos de red, que interviene durante todo el tiempo de ejecución de la herramienta de análisis dinámico, capturando los paquetes que se generan en la herramienta hacia la aplicación web y analizando cada uno de ellos en busca de patrones que sugieran el intento de un ataque hacia cualquiera de las vulnerabilidades que contenga la aplicación web.

En este modelo se generan 2 informes distintos que deben ser analizados:

- **Informe de vulnerabilidades halladas en la aplicación web**: Generado por la herramienta de análisis dinámico utilizada.
- **Informe de alertas de ataque**: Generado por el IDS durante la ejecución de la herramienta, presenta las alertas de ataque identificadas.

Analizando de esta forma las peticiones realizadas por las herramientas de análisis se puede obtener información relevante sobre las pruebas, adicional a la que se obtiene por el método tradicional. La información nueva que se obtiene es, en primer lugar, si las herramientas prueban todas las vulnerabilidades en la aplicación para las que tiene capacidad de detección y, en segundo lugar, si informa de todas las vulnerabilidades existentes que han probado. Es decir, puede que las herramientas dispongan de funcionalidades para la detección de XSS, la aplicación sea vulnerable a XSS, pero ni siquiera lo pruebe. También puede darse el caso de que la herramienta llegue a ejecutar esta prueba, pero que finalmente no informe de ello, aunque la herramienta sea vulnerable.

En la Figura 6.8 se puede observar el flujo de trabajo dentro del modelo propuesto.

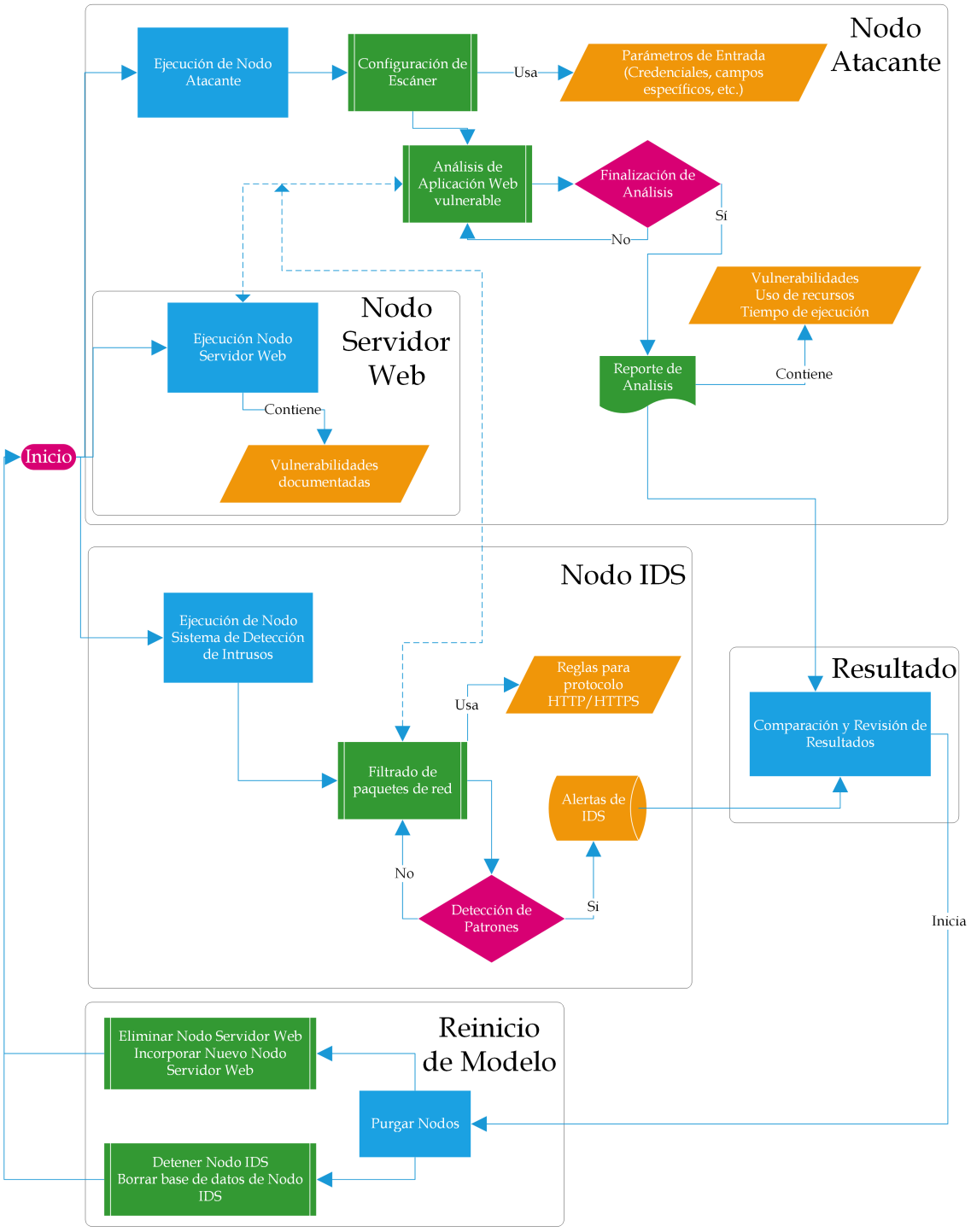


Figura 6.8: Diagrama de modelo propuesto

6.4. Síntesis del Capítulo

El objetivo de este capítulo ha sido presentar los problemas actuales de las herramientas automáticas de análisis de vulnerabilidades en aplicaciones web y las propuestas realizadas para solucionar o al menos mitigar cada uno de ellos. Estos problemas se resumen en:

1. Tratamiento inapropiado de los formularios web, principalmente con campos con restricciones fuertes y ejecución incorrecta del código cliente, como el JavaScript.
2. No disponer de una lista única de vulnerabilidades que deberían detectar, o al menos intentarlo.
3. No disponer de una aplicación vulnerable con el mayor número de vulnerabilidades implementadas.
4. Desconocimiento de las pruebas que realmente hacen sobre las aplicaciones vulnerables.

Para el primero se propone un algoritmo de preproceso de formularios web para obtener automáticamente valores de los campos, que puedan usarse en las siguientes fases del análisis de vulnerabilidades de una aplicación web. Estos valores serán más adecuados que los que puedan tener por defecto las herramientas actuales en dos aspectos. En primer lugar, cumplirán mejor con las restricciones sobre el tipo de campo impuestas por la aplicación que se esté analizando. En segundo lugar, estarán relacionados entre ellos correctamente (porque estén relacionados en una Tabla de Fusión o porque se haya ejecutado el código AJAX asociado al campo para obtenerlos, como lo haría un usuario con su navegador).

Para el segundo problema se propone un método para obtener una clasificación actualizada de vulnerabilidades en aplicaciones web que incluya las relaciones entre las clasificaciones existentes. Las relaciones que hay hasta la fecha entre clasificaciones son bastantes estáticas y en este trabajo se progresa en la dirección de poder tener actualizadas las relaciones entre las clasificaciones con la información disponible en cada momento.

Para el tercer problema se ha propuesto la selección de un conjunto de aplicaciones vulnerables, que contenga el mayor número de vulnerabilidades, sobre las que se añadirán vulnerabilidades adicionales no implementadas en las aplicaciones vulnerables actuales.

En el último problema, el enfoque propuesto incluye el uso de una herramienta auxiliar, un IDS, que se situará entre las herramientas y las aplicaciones analizadas, de forma que sea posible determinar qué pruebas, de entre las que puede hacer y las que debe hacer cada herramienta, son las que realmente hacen.

Capítulo 7

Preproceso de Formularios en Aplicaciones Web

En este capítulo se desarrollan las técnicas y los algoritmos definidos para la solución de la principal debilidad de la fase pasiva de las herramientas de análisis automático de aplicaciones web. Comienza con la descripción de la solución desarrollada, indicando las tareas que realiza cada uno de los módulos de los que se compone esta solución, y cómo se relacionan entre ellos. Seguidamente se detallan las pruebas realizadas para comprobar la solución sobre varios formularios de aplicaciones web. El capítulo finaliza con una breve síntesis de lo expuesto en el mismo.

7.1. Implementación de la Solución

Para realizar el preprocesado de los formularios de las aplicaciones web y obtener un conjunto de valores correctos que se puedan usar durante la fase de rastreo, se ha desarrollado un programa software. Se ha utilizado el lenguaje de programación Ruby empleado actualmente en varias herramientas de análisis de vulnerabilidades y que incorpora librerías específicas para la navegación y análisis de páginas web. Este programa se ejecuta sobre los formularios de las aplicaciones a analizar, como fase previa al rastreo. En la Figura 7.1 se muestra la situación de esta nueva fase dentro del flujo del análisis dinámico.

Con los valores encontrados en esta nueva fase se podrán llegar a más zonas de la aplicación que de otra forma no sería posible localizar. El programa consta de varios módulos, cada uno de los cuales se encarga de una de las tareas del preprocesado:

1. El primer paso es analizar el código HTML, para obtener los formularios y los campos de cada uno. Para cada formulario se registra su nombre, la acción (la URL a donde se envían los valores de los campos), el método (normalmente GET o POST) y el tipo de contenido (codificación de los caracteres del contenido que se envía). Para los campos su tipo (selección, texto libre, casilla de verificación, contraseña, etc.), su número de orden en el formulario y si está oculto o no. Si el campo está oculto el navegador no lo mostrará al usuario.

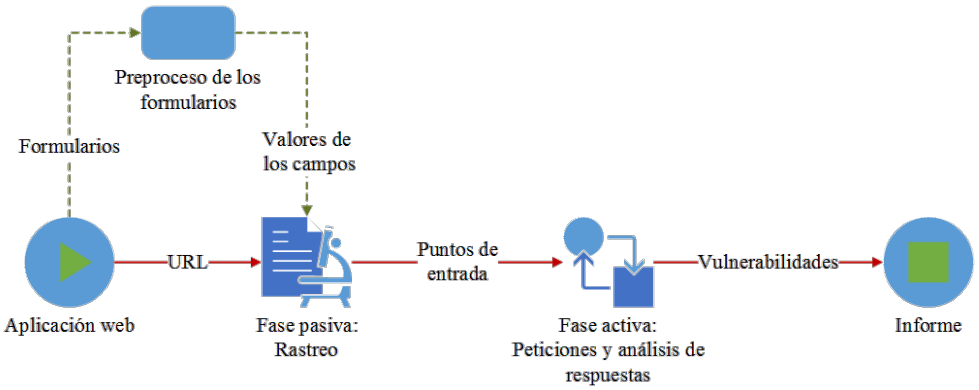


Figura 7.1: Preproceso de formularios en el análisis dinámico

2. Aunque es una práctica recomendada, no es necesario poner los literales que se muestran en pantalla asociados a los campos iguales que los nombres de los campos. Además, los literales en pantalla sí son descriptivos y entendibles por un usuario, pero los nombres de los campos muchas veces no lo son. Por ello es necesario obtener, para cada campo, el literal asociado que se muestre en la pantalla del navegador. Para realizar esta tarea se dispone de un módulo encargado de analizar la estructura del DOM de la página y extraer esos literales.

3. Con la información del código HTML y de la estructura del DOM, hay otro módulo que se encarga de comparar código y estructura, para obtener el literal más cercano, tanto en distancia como en parecido.

4. También hay un módulo encargado de encontrar los valores de los campos, con los que se probará el formulario. Si el campo tiene ya definido algún valor por defecto, campos de selección o casillas de verificación, será ese valor el que se guarde para probar. Si no tiene valores por defecto, pero se pueden seleccionar, se probará con ellos. Para el resto hay dos casos distintos. El primer caso es que sea un campo de selección, pero que la página no tenga valores, ya que se recuperan mediante peticiones AJAX, al elegir una opción en un campo anterior. En este caso este módulo, que busca los valores de los campos, abre un navegador web, rellena los campos de selección y espera la respuesta de las peticiones que devuelvan los valores de estos campos de selección para los que al principio no había valores.

5. El segundo caso es que no haya valores, al ser campos de texto libre. En este caso busca una tabla de Fusión de Google que pueda tener valores para él. Se busca primero una Tabla de Fusión que tenga una columna con nombre del campo; si no se encuentra ninguna, se busca una que tenga el literal del DOM más cercano según la distancia de Levenshtein; en tercer lugar, el literal más cercano del DOM, y, por último, se buscan sinónimos del nombre del campo. De la tabla que se recupera se extrae una cantidad de valores, cinco en principio, con los que probar.

En esa tabla de fusión que se encuentre, también se buscarán los demás campos de los que no se tengan valores, antes de buscar una tabla nueva para ellos. Por ejemplo, si hay que buscar valores para un país y un código postal de ese país, si se encuentran valores para los dos campos en la misma fila de una tabla, serán posiblemente valores buenos y que estén asociados. Será más probable que el formulario acepte, estos dos valores que si se han encontrado por separado.

6. Con todos los valores recuperados se prueba el formulario hasta que se encuentre un conjunto de valores que la aplicación acepte, o se quede sin valores con los que probar.

7. Para determinar si la aplicación ha aceptado los valores, el programa comprueba que la aplicación no devuelva un error y que haya cambiado de página. Si se permanece en la misma página será porque no todos los valores son correctos. En este último caso, se recuperan los valores de los campos que se mantienen rellenos, por lo que serán correctos y se prueba con nuevos valores en los campos que devuelve vacíos.

En la Figura 7.2 se muestran los módulos del programa desarrollado y las relaciones entre ellos.

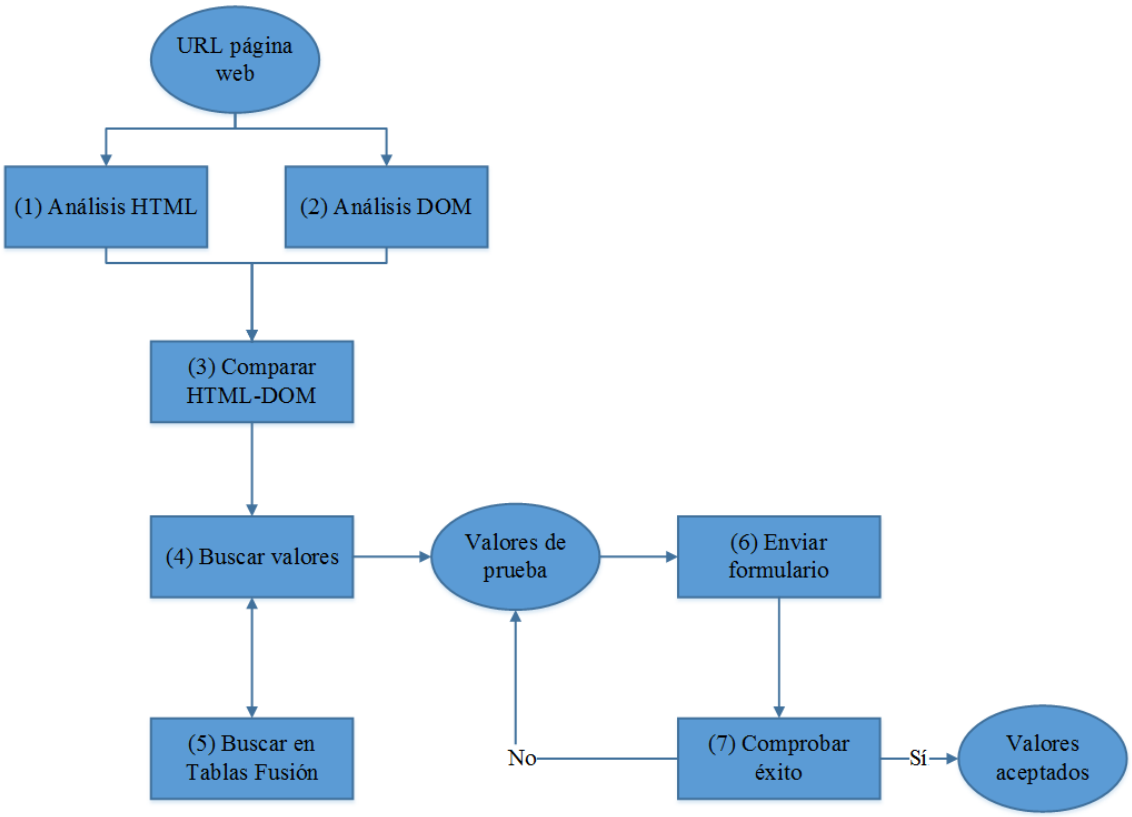


Figura 7.2: Fases del preproceso de formularios web

7.2. Prueba del Algoritmo

Para comprobar si el algoritmo propuesto proporciona mejores resultados que las herramientas actuales en la tarea de rellenar formularios web, se realizan dos tipos de pruebas.

1. En primer lugar, se realiza una prueba masiva sobre un conjunto de páginas web, para comprobar si los valores obtenidos en el preproceso de los formularios pueden ser válidos, es decir, aceptados por la aplicación.

2. En segundo lugar, se prueba el programa completo en un formulario de alta de usuarios de la tienda virtual Opencart [Ope13].

La primera prueba se realiza con las siguientes características:

a. Se toman 50 de las páginas web más visitadas.

b. Se obtienen los formularios y los campos (tipo, valores posibles, textos asociados, etc.).

c. Se extraen los valores existentes para los campos que los proporcionen, como los campos de selección, si es necesario ejecutando previamente el código de cliente.

d. Se buscan valores para los campos de texto en las Tablas de Fusión de tres formas diferentes, siempre buscando el nombre del campo y el texto del DOM si para el anterior no se encuentra nada:

   d.1 Cada campo de cada una de las 50 páginas diferentes por separado.

   d.2 Cada par de campos de cada una de las 50 páginas diferentes por separado.

   d.3 La primera vez que aparezca un campo. El valor que se obtenga se usará después como valor por defecto de ese campo.

e. Se establece el resultado obtenido con Burp Suite como línea base.

f. Se envían los formularios con los valores obtenidos para los campos tanto de selección como de texto de la siguiente forma:

   f.1 Únicamente los valores proporcionados por los propios campos (valores por defecto).

   f.2 Los valores por defecto más los valores encontrados por campo y página.

   f.3 Los valores por defecto más los valores encontrados por campo y página y, en caso de no encontrarlo, el valor que se ha encontrado para la primera página donde estuviera. Puede ser, por ejemplo, que un nombre de campo de texto libre anteriormente se haya encontrado en un campo de selección, por lo que se tendrán valores.

   f.4 El mismo caso anterior, pero utilizando un valor por defecto (“1”) en los campos que queden sin valor.

f.5 El caso anterior, pero antes de nada tomar los valores de las búsqueda en las Tablas de Fusión de dos en dos.

En la Tabla 7.1 se muestran los resultados de estas pruebas, en las que se indica el porcentaje de páginas nuevas que se encuentra en cada caso, en comparación con las que se encuentra con la herramienta Burp Suite realizando la fase de rastreo sobre esas mismas páginas y utilizando sus valores por defecto para los campos. Como se ve es en el último caso en el que se logra encontrar un número mayor de páginas de las aplicaciones, que Burp Suite no es capaz de encontrar, al buscar valores para los campos de texto de dos en dos. Esto se debe a que, al encontrar una Tabla de Fusión con los dos campos, los valores que contenga posiblemente estén realmente relacionados, como un código postal con su país, o una población con su provincia.

Tabla 7.1: Porcentaje de nuevas páginas encontradas

Opción	**f.1**	**f.2**	**f.3**	**f.4**	**f.5**
% páginas nuevas	7,94	13,64	13,81	19,42	22,18

En el segundo experimento realizado se prueba el programa desarrollado contra el formulario de alta de usuarios de la tienda virtual OpenCart. En este caso también se comparan los resultados con los que da la herramienta de uso habitual Burp Suite rellenando los campos con los valores que tiene esta herramienta por defecto. En la Figura 7.3 se muestra el formulario relleno con esos valores por defecto de Burp Suite. Como se ve el formulario no avanza ya que hay muchos de ellos que no los considera válidos; por ejemplo, la dirección, el código postal o la región, para la que previamente hay que hacer una llamada al servidor por AJAX.

Al analizar el formulario con el programa desarrollado se obtienen valores para cada uno de los campos de acuerdo a la Tabla 7.2, donde se indica el nombre del campo, su tipo y subtipo, y de dónde se han obtenido los valores.

Tabla 7.2: Tipos de campo y origen de los valores en el formulario de prueba

Nombre del campo	**Tipo de campo**	**Subtipo de campo**	**Origen de los valores**
Firstname (C1)	Texto	Texto	Tabla de fusión
Lasname (C2)	Texto	Texto	Tabla de fusión
Email (C3)	Texto	Texto	Tabla de fusión
Address (C4)	Texto	Texto	Tabla de fusión
Password (C5)	Texto	Contraseña	Calculado
Confirm (C6)	Texto	Contraseña	Calculado
Country_id (C7)	Selección	Desplegable	Página web
Zone_id (C8)	Selección	Desplegable	Página web
Newsletter (C9)	Selección	Botón de selección	Página web

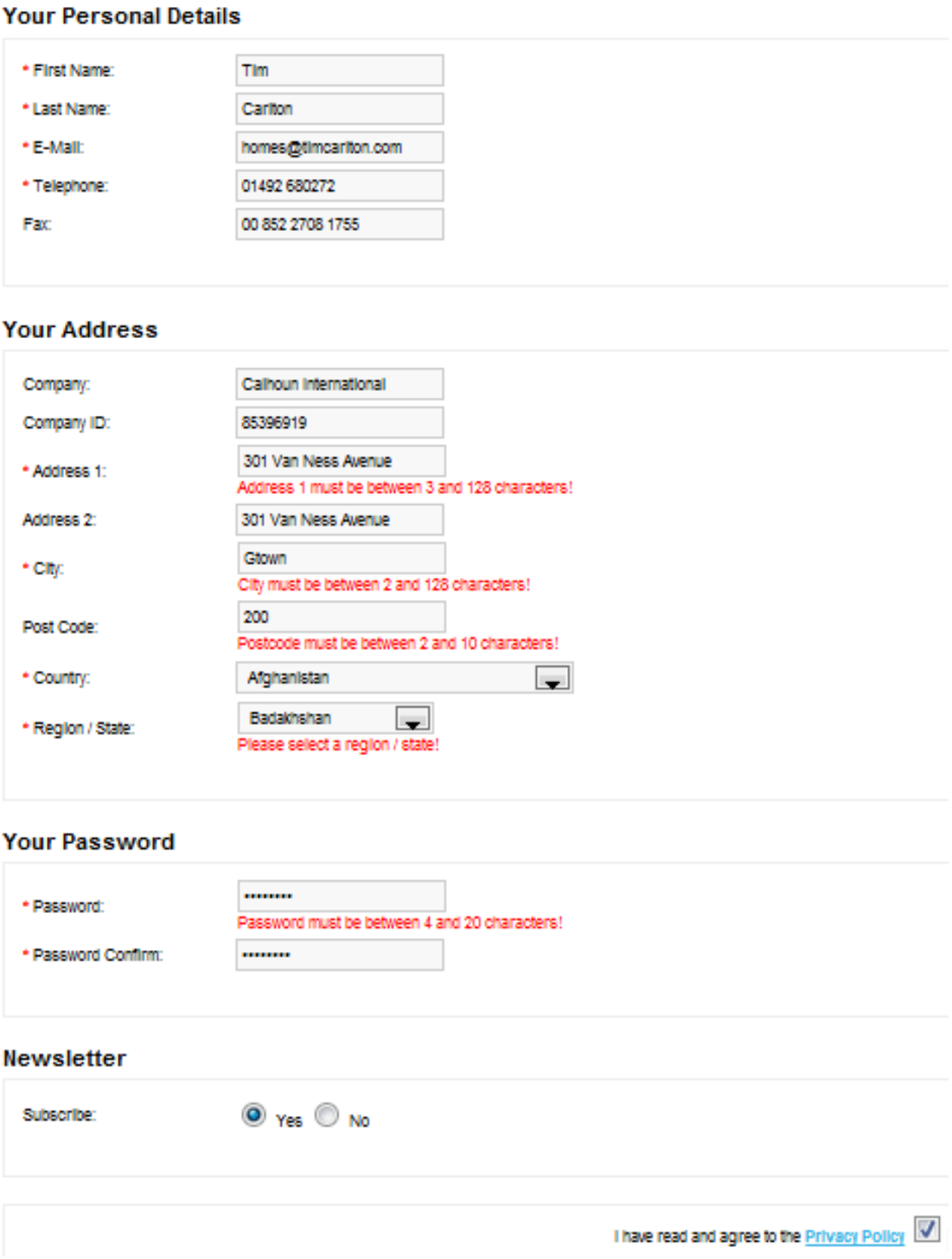


Figura 7.3: Formulario relleno con los valores por defecto de Burp Suite

El programa realiza varias búsquedas de valores y pruebas del formulario hasta que finalmente da con un conjunto de valores que la aplicación acepta y le permite seguir a la siguiente fase del flujo de alta de usuario. En cada iteración el programa envía los valores que tiene en ese momento. Estas iteraciones se muestran en la Tabla 7.3.

Como se ve en la tabla hay varios valores (como el C1, C2 y C4) que encuentra en la misma Tabla de Fusión al buscar el primero de ellos y, que como se esperaba, están relacionados.

Tabla 7.3: Iteraciones de la búsqueda de valores correctos

Iteración	Campo que buscar	Campos que encuentra	Campos con valores que envía
1	C1	C1, C2 y C4	C1, C2 y C4
2	C3	C3	C1 al C4
3	C5	C5 y C6	C1 al C6
4	C7	C7	C1 al C7
5	C8	C8	C1 al C8
6	C9	C9	C1 al C9

Como se muestra en la Figura 7.4, con seis iteraciones el programa ha sido capaz de encontrar valores correctos para los nueve campos del formulario y se ha conseguido dar de alta el usuario.

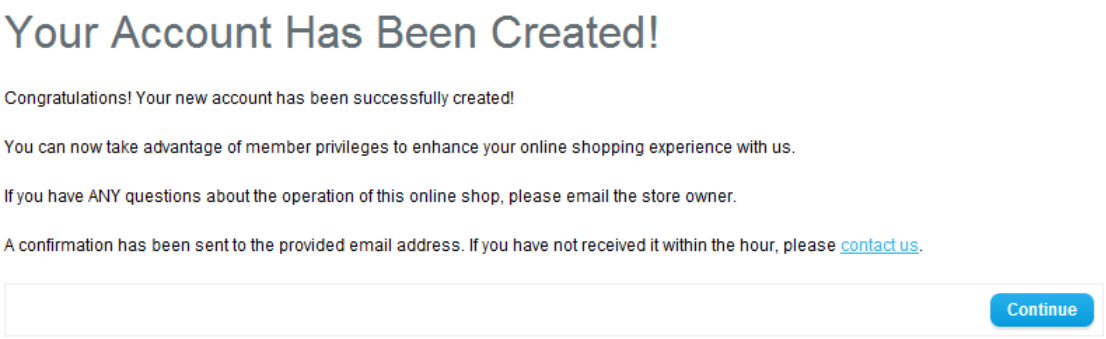


Figura 7.4: Alta de usuario

7.3. Síntesis del Capítulo

El objetivo de este capítulo ha sido presentar la implementación y prueba del algoritmo definido para el preprocesado de formularios de aplicaciones web. El objetivo del preproceso es obtener, para los campos de los formularios, valores que la aplicación acepte y permita continuar en el flujo definido en ella. Se ha comenzado con la exposición del programa software desarrollado para realizar la búsqueda de valores válidos, describiendo las tareas que realiza cada uno de los módulos que lo componen. Se ha continuado con los dos tipos de pruebas que se han hecho del algoritmo: una primera para comprobar si los valores que se obtendrían con él serían capaces de encontrar más páginas que una herramienta de uso habitual, y otra segunda para probar el programa sobre un formulario y determinar los detalles de la prueba.

Como se ve, el método propuesto permite localizar más páginas que las herramientas habituales, y fijándose en el detalle, encuentra valores válidos para campos que esas herramientas no son capaces de encontrar.

Capítulo 8

Lista Completa de Vulnerabilidades Web

En este capítulo se desarrolla el algoritmo definido para la obtención de una lista completa de vulnerabilidades en aplicaciones web, que unifica las clasificaciones ya existentes y que puede usarse para determinar las capacidades de detección de las herramientas automáticas. Comienza describiendo los pasos seguidos para obtener una clasificación de vulnerabilidades que incluya las que se encuentran en las clasificaciones más relevantes y conocidas, relacionándolas entre ellas. Seguidamente se indican las pruebas realizadas para comprobar si las relaciones encontradas entre las vulnerabilidades de distintas aplicaciones son ciertas. El capítulo finaliza con una breve síntesis de lo expuesto en el mismo.

8.1. Clasificación Completa de Vulnerabilidades Web

Para obtener una clasificación lo más completa posible de vulnerabilidades en aplicaciones web se ha seguido el algoritmo descrito en un apartado anterior. Siguiendo esos pasos se tiene:

1. Seleccionar el conjunto de clasificaciones de aplicaciones web que se va a tener en cuenta. Para esta fase se han considerado las siguientes: las guías de pruebas de OWASP, OWASPTG, en sus versiones OWASPTGv3 y OWASPTGv4, la versión 2.0 de la clasificación de amenazas en aplicaciones web de WASC, WASCTC, y la clasificación Common Weakness Enumeration (CWE). De la clasificación CWE sólo se han tenido en cuenta las vulnerabilidades web. De los posibles motores de búsqueda en Internet para realizar las búsquedas se ha probado con Bing y con Google, siendo este último el seleccionado ya que arrojaba un mayor número de posibles resultados válidos.

2. Localizar las relaciones entre clasificaciones que proporcionan los propios organismos que elaboran las clasificaciones. En este caso se tiene el que proporciona WASC que relaciona las vulnerabilidades en su clasificación WASCTC con las de CWE. También OWASP da información sobre las relaciones entre su última clasificación OWASPTGv4 y la versión anterior OWASPTGv3. En este

caso OWASP no proporciona realmente un mapeo, sino que en su sitio web, para cada vulnerabilidad de su clasificación OWASPTGv3 se referencia, si existe, la página correspondiente a la vulnerabilidad en la nueva clasificación. Por ejemplo, realizando la siguiente consulta en el motor de búsqueda "q=site:owasp.org+'Redirect+page'+-Category+'OWASP-DV-001"' se obtiene una página que enlaza "OWASP-DV-001" de OWASPTGv3 con "OTG-INPVAL-001" de OWASPTGv4. Para encontrar estas relaciones se ha implementado un proceso automático mediante el cual se consigue obtener la vulnerabilidad de OWASPTGv4 asociada a cada vulnerabilidad de OWASPTGv3, en caso de que OWASP proporcione esa información.

3. Añadir las relaciones que puedan proporcionar otras organizaciones de confianza. En este caso, una vez analizadas las relaciones indicadas en la Tabla 3.4, se han seleccionado las desarrolladas por Telligent y Sectoolmarket, ya que son los que proporcionan relaciones entre vulnerabilidades de las clasificaciones seleccionadas. Tanto Telligent como Sectoolmarket relacionan las pruebas de seguridad de OWASPTGv3 con las amenazas de WASCTC. Aunque la guía de pruebas de OWASP ya va por su versión v4 no ha sido posible encontrar ningún mapeo que la relacionara con alguna otra clasificación. Uno de los resultados del trabajo que se describe en este capítulo es precisamente este mapeo al relacionarla con las otras tres consideradas.

4. Reducir las descripciones de cada vulnerabilidad a un conjunto de palabras pequeño y único por vulnerabilidad, que a continuación se usa para buscar relaciones adicionales. Para realizar el proceso de reducir las descripciones de las vulnerabilidades a unas pocas palabras se ha desarrollado un programa software. Inicialmente se ha ejecutado el programa para reducir las descripciones de las vulnerabilidades a dos palabras, pero el resultado no era el esperado ya que se llegaba a una situación en la que más de una vulnerabilidad quedaba reducida al mismo grupo de palabras. Ejecutando el programa una segunda vez, buscando reducir cada descripción a tres palabras, el resultado ha sido el esperado y para cada vulnerabilidad se ha obtenido un grupo distinto de tres palabras.

5. Conseguir nuevas relaciones entre vulnerabilidades; para ello se ha implementado un proceso automático que compone búsquedas, las envía y analiza el resultado, para encontrar en las páginas de las organizaciones los conjuntos de tres palabras y los prefijos de vulnerabilidad de cada clasificación. Por ejemplo, enviando la consulta "q=site:wasc.org+'WASC-'ssl+tls+ciphers"' se obtiene la relación de la vulnerabilidad asociada a las palabras "ssl", "tls" y "ciphers" con WASC-04.

En la Tabla 8.1 se muestran varios ejemplos del cuarto paso. En la primera columna de esta tabla se indica el código de la vulnerabilidad en las cuatro clasificaciones seleccionadas; en la siguiente columna se muestra el nombre o descripción de la vulnerabilidad asociado a cada código; en la tercera columna lo que figura es el resultado de juntar todas las palabras en los nombres o descripciones asociados a esos códigos y a continuación eliminar todas las palabras repetidas y otros caracteres como paréntesis y comillas. En la última columna

figuran las tres palabras que resumen esa vulnerabilidad. Cada conjunto de tres palabras es diferente para cada vulnerabilidad y se puede considerar como las etiquetas que la definen. El resultado de esta búsqueda de relaciones entre vulnerabilidades web de diferentes clasificaciones y el conjunto mínimo de palabras que describe cada vulnerabilidad, se ha plasmado en una página web desarrollada para ello.

Tabla 8.1: Reducción a tres palabras de varias vulnerabilidades

Vulnerabilidad	Palabras por vulnerabilidad	Sin repeticiones	3 palabras
OWASP-CM-004 OTG-CONFIG-002 WASC-14 CWE-16	*Application Configuration Management Testing Test Application Platform Configuration Server Misconfiguration Configuration*	*Application Configuration Management Testing Test Platform Server Misconfiguration*	*platform server misconfiguration*
OWASP-AT-004 OTG-AUTHN-003 WASC-11 CWE-350	*Brute Force Testing Testing for Weak lockout mechanism Brute Force Predictability Problems*	*Brute Force Testing Weak lockout mechanism redictability Problems*	*lockout predictability problems*
OWASP-AZ-001 OTG-AUTHZ-002 WASC-33 CWE-22	*Testing for Path Traversal Testing for bypassing authorization schema Path Traversal Improper Limitation of a Pathname to a Restricted Directory: (‘Path Traversal’)*	*limitation pathname restricted Testing Path Traversal bypassing authorization schema Improper* textitRestricted Directory	*limitation pathname restricted*

En la Figura 8.1 se muestra una pantalla de esta página con un ejemplo de las relaciones obtenidas a partir de las relaciones previas.

You have selected: OTG-CLIENT-001

-Column **Words** have three-words vulnerability summaries.
-Green is used for relationships between OWASP testing guide v3 and v4 given by OWASP, and for relationships between WASC v2 and CWE given by MITRE.
-Relationships between OWASP testing guide and WASC v2 or CWE are given by external sources like Telligent or Sectoolmarket.

Words	OWASP v3	OWASP v4	WASC v2	CWE
dom based page	OWASP-DV-003 Testing for DOM based Cross Site Scripting	OTG-CLIENT-001 Testing for DOM based Cross Site Scripting	WASC-08 Cross-site Scripting	CWE-79 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

Figura 8.1: Resultado de aplicar el método descrito para conseguir relaciones existentes entre las vulnerabilidades de las clasificaciones seleccionadas

En la Figura 8.2 se muestra un ejemplo de las nuevas relaciones obtenidas a partir del conjunto mínimo de palabras de cada vulnerabilidad.

3.- 2nd order relationships

You have selected: OTG-BUSLOGIC-001

-Orange are for relationships given by Google searching on classification developer sites.

Code	Vulnerability
CWE-105	Struts: Form Field Without Validator
CWE-107	Struts: Unused Validation Form Weakness_Abstraction="Variant"
CWE-129	Improper Validation of Array Index Weakness_Abstraction="Base"
CWE-172	Encoding Error Weakness_Abstraction="Class" Status="Draft">
CWE-173	Improper Handling of Alternate Encoding
CWE-20	Improper Input Validation
CWE-21	Pathname Traversal and Equivalence Errors
CWE-840	Business Logic Errors
CWE-841	Improper Enforcement of Behavioral Workflow
WASC-40	Insufficient Process Validation

Figura 8.2: Resultado de aplicar el método descrito para conseguir nuevas relaciones entre las vulnerabilidades de las clasificaciones seleccionadas

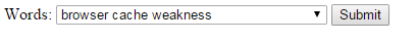
En ella, seleccionado un código de vulnerabilidad de alguna de las cuatro clasificaciones que se han tenido en cuenta, se muestra:

- Conjunto de tres palabras que describe la vulnerabilidad.
- Vulnerabilidades de las otras clasificaciones con las que está relacionada a partir de las relaciones de las propias organizaciones o de terceros para las clasificaciones para las que exista alguna de estas relaciones.
- Vulnerabilidades con las que está relacionada, obtenidas a partir de consultas en Internet en las páginas de las organizaciones que elaboran las clasificaciones.

Como ya se ha mencionado anteriormente, uno de los resultados de este trabajo es el desarrollo de nuevas relaciones entre clasificaciones de vulnerabilidades que hasta ahora no existían. En concreto, se obtiene un mapeo entre OWASPTGv4 y WASCTC y otro entre OWASPTGv4 y CWE. También se actualiza el mapeo entre WASCTC y CWE del que ya hay uno en Webappsec de 2013.

También se han desarrollado otras dos páginas para proporcionar nuevas relaciones entre vulnerabilidades de diferentes clasificaciones. Estas dos páginas hacen consultas en tiempo real. En la primera se pueden buscar relaciones en tiempo real en los sitios web de las organizaciones que elaboran las clasificaciones, y en la segunda en todo Internet, no sólo en esos sitios web.

La información que proporcionan, al ser en tiempo real, podrá cambiar de una consulta a otra. Incluirá nuevas relaciones a partir de nueva información en Internet y otras relaciones podrán desaparecer de una consulta a la siguiente. Aunque esta información quizás no sea tan fiable como la que puedan dar las organizaciones, sí aporta valor a la hora de buscar información sobre una vulnerabilidad y sus relaciones, ya que puede incluir el trabajo realizado por profesionales de la seguridad de la información que han tratado una vulnerabilidad en concreto. Las Figuras 8.3(a) y 8.3(b) son ejemplos de los resultados obtenidos en estas páginas.



You have selected: browser cache weakness

Number	Code	Vulnerability
0	OWASP-AT-007	Testing for Logout and Browser Cache Management
1	OTG-SESS-001	Testing for Bypassing Session Management Schema
2	OTG-SESS-007	Test Session Timeout
3	OTG-AUTHN-006	Testing for Browser cache weakness
4	OTG-IDENT-005	Testing for Weak or unenforced username policy
5	OTG-AUTHN-003	Testing for Weak lock out mechanism
6	OTG-CRYPST-001	Testing for Weak SSL/TLS Ciphers, Insufficient Transport Layer Protection
7	CWE-525	Information Exposure Through Browser Caching
8	CWE-524	Information Exposure Through Caching
9	CWE-113	Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting')
11	CWE-200	Information Exposure
12	CWE-312	Cleartext Storage of Sensitive Information
13	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
14	CWE-346	Origin Validation Error

(a) Relaciones halladas en sitios web de clasificaciones

You have selected: xpath injection testing

Number	Code	Vulnerability
0	OWASP-DV-010	XPath Injection
1	OWASP-WS-004	XML content-level Testing
2	OTG-INPVAL-010	Testing for XPath Injection
3	WASC-39	XPath Injection
4	WASC-28	Null Byte Injection
5	WASC-04	Insufficient Transport Layer Protection
6	WASC-23	XML Injection
7	WASC-43	XML External Entities XXE)
8	WASC-41	XML Attribute Blowup
9	CWE-91	XML Injection (aka Blind XPath Injection)
10	CWE-643	Improper Neutralization of Data within XPath Expressions ('XPath Injection')
11	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
12	CWE-77	Improper Neutralization of Special Elements used in a Command ('Command Injection')
13	CWE-315	Cleartext Storage of Sensitive Information in a Cookie

(b) Relaciones halladas en todo Internet

Figura 8.3: Ejemplo de relaciones encontradas con consultas en tiempo real

8.2. Prueba y Análisis de los Resultados

Una vez definido e implementado el método para relacionar las vulnerabilidades de las clasificaciones ya existentes y de esa manera obtener una clasificación unificada, en este apartado se prueba si realmente las relaciones obtenidas son ciertas.

8.2.1. Configuración de las Pruebas

Para probar si el método descrito e implementado en los apartados anteriores es adecuado, se seleccionan varias vulnerabilidades y se comprueba si las relaciones que se obtienen son relaciones ciertas. Las relaciones entre elementos de clasificaciones serán ciertas si son claramente la misma vulnerabilidad, o si las dos aportan información (prueba o resolución) de la misma vulnerabilidad. Las vulnerabilidades seleccionadas son cinco de las 10 que confirman la lista de riesgos OWASPT10 de 2013.

Estas cinco vulnerabilidades tienen de forma clara su correspondiente elemento en cada una de las cuatro clasificaciones consideradas. En la Tabla 8.2 se muestran estas vulnerabilidades, el resumen de tres palabras de cada una y el código en las clasificaciones.

Tabla 8.2: Vulnerabilidades seleccionadas para probar los resultados de implementar el algoritmo descrito

Nombre en inglés	Resumen de tres palabras	OWASPTGv3	OWASPTGv4	WASCTC	CWE
SQLI	Testing SQL Injection	OWASP-DV-005	OTG-INPVAL-006	WASC-19	CWE-89
Session fixation	Testing Session Fixation	OWASP-SM-003	OTG-SESS-003	WASC-37	CWE-384
RXSS	Reflected Page Generation	OWASP-DV-001	OTG-INPVAL-001	WASC-08	CWE-79
Path trasversal	Limitation Path Restricted	OWASP-AZ-001	OTG-AUTHZ-002	WASC-33	CWE-22
CSRF	CSRF Request Forgery	OWASP-SM-005	OTG-SESS-005	WASC-09	CWE-352

8.2.2. Resultado de las Pruebas

Buscando en las páginas desarrolladas las cinco vulnerabilidades indicadas, se encuentran las relaciones que se muestran en la Tabla 8.3.

Tabla 8.3: Relaciones entre las cinco vulnerabilidades seleccionadas

Resumen de tres palabras	**Relaciones de 1er orden**	**Relaciones de 2° orden estáticas**	**Relaciones de 2° orden dinámicas**	**Relaciones de 3er orden (dinámicas)**
testing sql injection	OWASP-DV-005 OTG-INPVAL-006 WASC-19 CWE-89	OWASP-DV-005, OTG-INPVAL-005 OTG-INPVAL-006, OTG-INPVAL-007 OTG-AUTHN-004, OTG-INPVAL-010 OTG-INPVAL-011, WASC-19 WASC-13, CWE-89, CWE-564 CWE-120, CWE-94, CWE-20, CWE-79, CWE-209, CWE-565 CWE-78	OWASP-DV-005, OWASP-AJ-001 OWASP-WS-004, OTG-INPVAL-005 OTG-INPVAL-006, OTG-INPVAL-007 OTG-AUTHN-004, OTG-INPVAL-010 OTG-INPVAL-011, WASC-19, WASC-13 CWE-89, CWE-564, CWE-120, CWE-94, CWE-20, CWE-79, CWE-565, CWE-209, CWE-78	OWASP-DV-005 OWASP-AJ-001 OTG-INPVAL-005 OTG-INPVAL-006 OTG-INPVAL-007 WASC-19, WASC-13, CWE-89, CWE-564 CWE-120, CWE-77 CWE-90
testing session fixation	OWASP-SM-003 OTG-SESS-003 WASC-37 CWE-384	OWASP-SM-003, WASC-37 WASC-47, CWE-384, CWE-664, CWE-732, CWE-287, CWE-698, CWE-79, CWE-352	OWASP-SM-003, OWASP-AT-009 OTG-SESS-003, OTG-SESS-001 WASC-37, WASC-47, CWE-384, CWE-664, CWE-732, CWE-287, CWE-698, CWE-79, CWE-352	OWASP-SM-003 OTG-SESS-003 OTG-SESS-004 WASC-18, WASC-37, WASC-47, CWE-384
reflected page generation	OWASP-DV-001 OTG-INPVAL-001 WASC-08 CWE-79	OTG-INPVAL-001, OTG-INPVAL-002, OTG-CLIENT-009, CWE-79, CWE-80, CWE-81, CWE-416	OWASP-DV-001, OTG-INPVAL-001, OTG-INPVAL-002, OTG-CLIENT-009 CWE-79, CWE-80, CWE-81, CWE-416	OWASP-DV-001 OTG-INPVAL-001 OTG-INPVAL-002 CWE-79, CWE-22 CWE-78, CWE-639
limitation path restricted	OWASP-AZ-001 OTG-AUTHZ-002 WASC-33 CWE-22	OWASP-AZ-001, OWASP-CM-001, OTG-CRYPST-003, OTG-CRYPST-001, WASC-33, CWE-22, CWE-494, CWE-36, CWE-23, CWE-41, CWE-21, CWE-119, CWE-73,	OWASP-AZ-001, WASP-CM-001, OTG-INFO-008, OTG-CRYPST-003, OTG-CRYPST-001, WASC-33, CWE-22, CWE-494, CWE-36, CWE-23, CWE-41, CWE-21, CWE-119, CWE-73	OWASP-AZ-001 OWASP-CM-001 WASC-04 WASC-33 CWE-22 CWE-494 CWE-23 CWE-264
CSRF request forgery	OWASP-SM-005 OTG-SESS-005 WASC-09 CWE-352	OWASP-AJ-001, OWASP-SM-005, OTG-SESS-005, CWE-352, CWE-441, CWE-442	OWASP-AJ-001, OWASP-SM-005, OTG-SESS-005, CWE-352, CWE-441, CWE-442	WASC-09 CWE-352

Para cada vulnerabilidad se da la siguiente información:

1. **Relaciones de primer orden**: Son las relaciones entre vulnerabilidades web que se obtienen a partir de relaciones entre las clasificaciones que han realizado las propias organizaciones que hacen las clasificaciones u otros organismos o empresas del ámbito de la seguridad de la información.

2. **Relaciones de segundo orden estáticas**: Son las relaciones que se obtuvieron cuando se implementó el algoritmo descrito, realizando consultas en Google sobre los sitios web de las organizaciones que desarrollan las clasificaciones, como OWASP, WASC o CWE.

3. **Relaciones de segundo orden dinámicas**: Son las relaciones que se obtienen realizando consultas en Google sobre los sitios web de las organizaciones que desarrollan las clasificaciones, pero que se realizan en tiempo real, en el momento de consultar la página web. En este caso la consulta se hizo en abril de 2015.

4. **Relaciones de tercer orden**: Son las relaciones entre vulnerabilidades que se obtienen realizando consultas también en tiempo real en Google sobre todo Internet.

8.2.3. Análisis de los Resultados

A partir de la Tabla 8.3 se pueden obtener algunas conclusiones:

- Las relaciones de primer orden, obtenidas de las organizaciones, coincide con las relaciones obvias de cada vulnerabilidad con las clasificaciones (ver Tabla 8.2). Este resultado es el esperado, al comparar las relaciones evidentes de cada vulnerabilidad con las relaciones que se elaboran.

- Aunque sólo han pasado dos meses entre las dos consultas para obtener relaciones de segundo orden, durante ese período aparecen nuevas relaciones y en un caso desaparece el código de vulnerabilidad “CWE-494” en “limitation path restricted”.

- En la mayoría de los casos las búsquedas de segundo y tercer orden incluyen las relaciones obvias en sus resultados (en verde en la Tabla 8.3), lo que apoya la bondad del método definido. En la Tabla 8.4 se ven los porcentajes de inclusión en cada caso.

- El resto de elementos que aparecen en cada caso o son la vulnerabilidad buscada o al menos incluyen información sobre los métodos de detección y de mitigación.

- Las relaciones obtenidas a partir de búsquedas en todo Internet son menores en número que las que proporcionan las búsquedas en los sitios web de las clasificaciones.

Tabla 8.4: Porcentaje de relaciones evidentes encontrada en cada tipo de búsqueda

Resumen de tres palabras	**Relaciones de 1er orden**	**Relaciones de 2° orden estáticas**	**Relaciones de 2° orden dinámicas**	**Relaciones de 3er orden (dinámicas)**
testing sql injection	100 %	100 %	100 %	100 %
testing session fixation	100 %	75 %	100 %	100 %
reflected page generation	100 %	50 %	75 %	75 %
limitation path restricted	100 %	75 %	75 %	75 %
CSRF request forgery	100 %	75 %	75 %	50 %
Media - 85 %	100 %	75 %	85 %	80 %

8.3. Síntesis del Capítulo

El objetivo de este capítulo ha sido presentar el trabajo de unificación de las clasificaciones de vulnerabilidades web y de obtención, por ello, de una clasificación que las contiene a todas ellas. Las clasificaciones actuales y sobre todo sus relaciones son bastante estáticas y no suelen actualizarse con frecuencia, aunque sí existe información en Internet sobre ellas y sus relaciones que se va actualizando constantemente. En este trabajo, además de la información que proporcionan los organismos que elaboran las herramientas, se utiliza esa información disponible en Internet que relaciona unas clasificaciones con otras y también únicamente unas pocas vulnerabilidades con otras. Después de describir el método implementado para mantener una clasificación y las relaciones actualizadas, se ha probado que esas relaciones obtenidas son ciertas utilizando un conjunto de vulnerabilidades bien conocidas y con relaciones obvias.

Capítulo 9

Aplicación Vulnerable a Propósito

En este capítulo se explican las características de la aplicación vulnerable a propósito y se prueba con un conjunto de herramientas de análisis. Comienza con la descripción de las vulnerabilidades adicionales que se implementan en las aplicaciones a propósito que ya tienen más vulnerabilidades. Seguidamente se describen las pruebas realizadas sobre esta nueva aplicación vulnerable con un conjunto de herramientas automáticas. A continuación, se indican los resultados de estas pruebas y las conclusiones que se pueden extraer de ellas. El capítulo finaliza con una breve síntesis de lo expuesto en el mismo.

9.1. Aplicación Vulnerable

Una vez que se tiene el conjunto de vulnerabilidades que deberían ser capaces de buscar las herramientas automáticas de análisis de vulnerabilidades en aplicaciones web, el siguiente paso es disponer de una aplicación vulnerable, o conjunto de aplicación, con el mayor número del vulnerabilidades implementada.

Para ello se seleccionan las dos aplicaciones vulnerables que actualmente ya contienen un mayor número de vulnerabilidades, Mutillidae II y WebGoat. Sobre la primera se han añadido 8 nuevos tipos de vulnerabilidades seleccionadas por su relevancia e información disponible. Las vulnerabilidades añadidas a Mutillidae II son las siguientes:

- **Abuso de funcionalidad, mecanismo de recuperación de contraseñas inseguro y proceso de validación insuficiente**: Estas vulnerabilidades pueden llevar a grandes pérdidas a las empresas por lo que son importantes detectarlas a tiempo. Se ha implementado como ejemplo de los errores que suelen cometerse en la lógica al desarrollar una aplicación.
- **Byte Nulo (Null Byte) e Inyección *Server-Side Include* (SSI)**: Aunque ya no son tan comunes, aún sigue habiendo aplicaciones con estas vulnerabilidades, por lo que es necesario implementarlas como ejemplo.
- **Inyección de código**: Se ha implementado debido a que los programadores suelen tener más en cuenta las inyecciones de SQL, pero no tienen en cuenta otro tipo de inyecciones como las de código de algún lenguaje como JavaScript o PHP.

- **Inyección XPath**: Se ha implementado debido a que muchas aplicaciones hacen uso de XPath para navegar a través de elementos y atributos de los documentos XML, pero no se hace una validación correcta por parte de los programadores.
- **Inyección NoSql**: Se ha implementado debido a que cada día las bases de datos NoSql se implementan más y no se toman las debidas precauciones en las validaciones de los datos de entrada.

En la Figura 9.1 se muestra una pantalla de las nuevas vulnerabilidades implementadas en Mutillidae II.

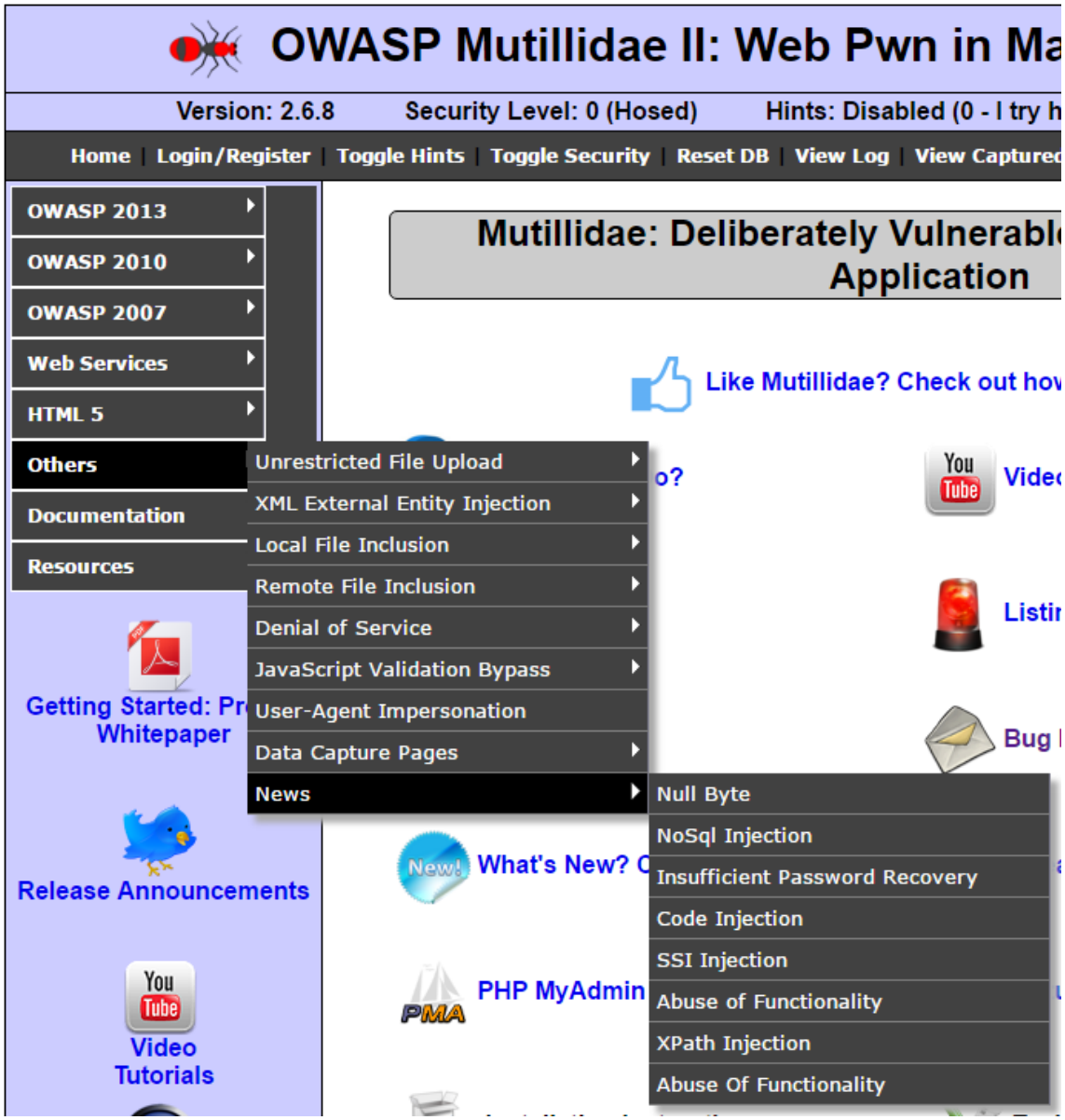


Figura 9.1: Nuevas vulnerabilidades añadidas a Mutillidae II

También hay un conjunto de vulnerabilidades que no se han implementado en la aplicación vulnerable porque necesitan de otros componentes o los propios entornos de desarrollo ya cuentan como mecanismos de protección frente a ellas. Algunas son:

- **Error de Desbordamiento de Enteros (*Integer Overflows*) y en el Formato de Cadenas (*Format String*)**: Los lenguajes de desarrollo cuentan con protecciones que hacen que sea difícil de implementar mediante ejemplos simples.
- **Contrabando de Peticiones HTTP (HTTP *Request Smuggling*) y Contrabando de Peticiones HTTP (HTTP *Response Smuggling*)**: No han sido implementadas debido a que se necesitan otros componentes como *proxies*, por lo que se deja como trabajo futuro.
- **Inyecciones de LDAP y de Comandos de Correo (*Mail Command Injection*)**: No han sido implementadas debido a que se necesita la instalación de servidores con estas tecnologías. Se deja como trabajo futuro.
- **División de Peticiones HTTP (HTTP *Response Splitting***: Las versiones modernas de Java y PHP cuentan con protección ante este ataque.
- **Desvío de Enrutamiento (*Routing Detour*), Abuso de *Array* SOAP y Vulnerabilidades en los Servicios Web**: Son ataques específicos de servicios web y no de aplicaciones web.
- **Inyección de XQuery**: No se ha implementado un ejemplo debido a que en PHP no se cuenta con un *parser* nativo. Se deja como trabajo futuro.
- **Inyección de Lenguaje de Expresión (*EL Injection*) y el Oráculo del Relleno (*Padding Oracle*)**: Por su complejidad se dejan como trabajo futuro para WebGoat.

9.2. Experimentos y Resultados

Se ha realizado un análisis de las aplicaciones vulnerables con las herramientas Vega [Sub16] versión 1.0 y OWASP ZAP [The16] versión 2.3, dos herramientas automatizadas para la búsqueda de vulnerabilidades web de código abierto muy conocidas. Se han utilizado dos aplicaciones web intencionalmente vulnerables para examinar las dos herramientas y comprobar sus capacidades de detección de vulnerabilidades. Para poder realizar el análisis con las herramientas se ha hecho lo siguiente:

1. Configurar y activar los distintos añadidos (*plugins*) de las herramientas para detectar vulnerabilidades.
2. Proporcionar a las herramientas la dirección de la aplicación para realizar un análisis automático.
3. Activar el *proxy* de la herramienta y realizar una navegación manual inicial de la aplicación web.

4. Verificar que las vulnerabilidades detectadas no sean falsos positivos.

Esto se ha realizado tanto para Mutillidae II como para WebGoat. El paso 2 se ha realizado de dos formas en ambas herramientas: (1) una navegación automática y, debido a la dificultad de las herramientas para realizar la navegación automática, (2) una navegación semiautomática en la cual se hizo una navegación manual con el *proxy* de la herramienta. Al finalizar la navegación se tomó como base todo lo capturado por el *proxy* y se realizó de nuevo el análisis automático.

En la Tabla 9.1 se observa la capacidad de detección de las herramientas Vega y OWASP ZAP. Vega ha sido la que más vulnerabilidades ha detectado, con 17 vulnerabilidades de las 49 que se encuentran en las dos aplicaciones. En Mutillidae II ha detectado 17 y 13 en WebGoat. También puede observarse que Vega sólo fue capaz de detectar una de las vulnerabilidades que fueron agregadas a Mutillidae II. Esta vulnerabilidad es la Inyección de Código. Mientras que OWASP ZAP fue capaz de detectar la vulnerabilidad CSRF, Vega no la ha detectado, siendo esta vulnerabilidad una de las más comunes actualmente. Por otro lado, Vega fue capaz de detectar Inyección de Código y Enumeración de Usuarios, al contrario que OWASP ZAP.

Tabla 9.1: Capacidades de detección de Vega y Zaproxy en las aplicaciones vulnerables

	Vega	**Zaproxy**
49 Vulnerabilidades (WebGoat y Mutillidae II)	17	15
Porcentaje de Detección	35.69	30.61
Mutillidae II	17	15
WebGoat	13	13
Vulnerabilidades Nuevas (Mutillidae)	1	0

Las vulnerabilidades que detectaron ambas herramientas se basan en la lista OWASPT10. Esto puede ser debido a que los desarrolladores de ambas herramientas basan sus pruebas en dicha lista, siendo ésta la más conocida y difundida entre los profesionales de la seguridad web. En la Tabla 9.1 también se puede observar que el porcentaje de detección es bajo. Esto es debido a la dificultad de las herramientas para realizar el rastreo y a los pocos *plugins* implementados para la detección de distintas vulnerabilidades, sobre todo, en tecnologías como NoSql, XML y JSON.

Durante las dos fases de análisis que se realizó con las herramientas, se pudo observar que la principal limitación de éstas es en la navegación dentro de la aplicación. Al realizar la navegación semiautomática se pudieron detectar más vulnerabilidades.

9.3. Síntesis del Capítulo

El objetivo de este capítulo ha sido presentar la ampliación con ocho vulnerabilidades, de las aplicaciones vulnerables que contienen un mayor número de ellas y, posteriormente, su prueba con varias herramientas automáticas actuales. Las aplicaciones consideradas se han elegido por tener claramente definidas las vulnerabilidades, ser fáciles de personalizar y de agregar nuevos tipos de vulnerabilidades, además de contar con tecnología y funcionalidad de aplicaciones web actuales, y estar bien documentadas.

Como puede observarse en los resultados, las herramientas sólo fueron capaces de detectar un porcentaje muy bajo del total, todas ellas de OWASPT10, y únicamente una de las nuevas vulnerabilidades incluidas.

Capítulo 10

Análisis de las Capacidades de las Herramientas

En este capítulo se muestra la solución para determinar las capacidades de detección de las herramientas automáticas de análisis de vulnerabilidades web. Comienza describiendo brevemente las herramientas que se han evaluado y de las que se quieren revisar sus capacidades de detección. Seguidamente se detalla el mecanismos propuesto para poder evaluar las capacidades de esas herramientas. A continuación, se indican y describen las aplicaciones vulnerables que se van a analizar. Se sigue con los resultados de las pruebas realizadas analizando las aplicaciones vulnerables con las herramientas. El capítulo finaliza con una breve síntesis de lo expuesto en el mismo.

10.1. Herramientas que se Evalúan

Para comprobar las capacidades reales de detección de las herramientas de análisis automático de vulnerabilidades en aplicaciones web se han seleccionado OWASP ZAP, Acunetix WVS, HP WebInspect y Arachni, ya descritas en detalle en un apartado anterior. Se han elegido estas herramientas debido a que son todas herramientas muy conocidas y utilizadas en el análisis de la seguridad de aplicaciones web, además de que en trabajos previos [ASW10] [Sae14a] [Sae14b] destacaron sobre el resto de herramientas con las que se las comparaba.

- **OWASP ZAP** es una herramienta gratuita de análisis dinámico de vulnerabilidades, parte del grupo de proyectos de la fundación OWASP, muy utilizado alrededor del mundo y que ofrece gran cantidad de documentación y soporte por parte de la comunidad de voluntarios que la mantiene y desarrolla. Permite realizar distintos tipos de análisis y ataques, permitiendo configurar perfiles específicos para ajustarlos a las características de las aplicaciones que se desean analizar. Para este trabajo se ha utilizado la versión 2.4.3 OWASP ZAP.

- **Acunetix WVS** es una herramienta comercial especializada en auditar aplicaciones web en busca de vulnerabilidades y fallos que puedan comprometer la integridad de la aplicación y la información que contiene. Es muy popular gracias a la gran cantidad de características que ofrece y que permiten configurar la herramienta para que el análisis mejore su precisión y obtener resultados más fiables en su informe. En este trabajo se ha utilizado la versión gratuita v10.5.

- **HP WebInspect** es una herramienta desarrollada por HP para el análisis dinámico de vulnerabilidades en aplicaciones web. La versión utilizada en este trabajo es la v10.5 de prueba gratuita que permite utilizar la herramienta con todas sus capacidades por 15 días. La configuración inicial requiere de poca aportación por parte del usuario y los informes que genera cuando ha terminado el análisis son muy completos.

- **Arachni** es una herramienta gratuita de análisis dinámico desarrollada en Ruby. Sus primeras versiones no presentaban una interfaz gráfica y por lo tanto la configuración y análisis de una aplicación se realizaba mediante el uso de la línea de comandos. La versión 1.4, utilizada en este trabajo, presenta una interfaz web que permite una fácil configuración de la herramienta y además logra de esta manera ser multiplataforma, pudiendo ejecutarse en entornos Windows, Linux y Mac OS X.

Todas las herramientas utilizadas en este trabajo fueron configuradas con los perfiles de análisis que traen por defecto, sin ajustar ninguna característica particular adicional, para que de esta manera el informe de análisis que se obtenga sea el que obtuviese un usuario sin conocimientos profundos sobre la herramienta.

Sin embargo, es importante indicar que cada una de las herramientas contiene añadidos y características que pueden ajustar la precisión del análisis o realizar un ataque a medida, en busca de un tipo de vulnerabilidad específica.

10.2. Mecanismos de Evaluación

Con el fin de obtener más información que permita comparar y evaluar las capacidades de las herramientas de análisis dinámico, se ha incluido un sistema de detección de intrusos del lado de la aplicación vulnerable, el cual genera un informe que se puede comparar con los que generan las herramientas de análisis dinámico. De esta forma se comprueba qué vulnerabilidad es explotada y se puede conocer si se analizan al menos las vulnerabilidades más conocidas y consideradas como las más importantes.

Para la evaluación de las herramientas de análisis se ha utilizado Snort, como IDS de red, configurado con setenta y tres mil reglas exclusivamente para tráfico HTTP/HTTPS. De esta manera se busca cubrir todas las posibilidades de ataque que cada herramienta realice durante su análisis.

Para gestionar todas las alertas y poder obtener un informe claro y fácil de analizar, se utilizó conjuntamente con Snort, Barnyard2 [Fir16] y Snorby [Wil16]. De esta manera se centralizan todas las alertas en una base de datos y se visualizan de forma clara a

través de un navegador web. En la Tabla 10.1 se muestra un detalle de las características de Snort y de las herramientas que fueron necesarias para procesar sus alertas.

Tabla 10.1: IDS y Componentes

	Snort	**Barnyard2**	**Snorby**
Versión	2.9.8.0	2-1.14	2.6.3
Desarrollador	Sourcerfire, Inc.	Ian Firns	Dustin Willis Webber

10.2.1. Herramientas de Administración

Para monitorizar el uso de recursos de red durante el proceso de análisis que realizaron cada una de las herramientas evaluadas se utilizó Darkstat. Darkstat es una herramienta que permite capturar el tráfico de red y obtener estadísticas de uso gráficos de tráfico e informes por *host*. Muestra en tiempo real el uso que se está efectuando de la red, especificando *host* y puertos de destino y fuente en cada caso. Tiene soporte para IPv6 y es de código abierto.

10.3. Aplicaciones Vulnerables

Como aplicaciones vulnerables se han utilizado en este caso DVWA y WackoPicko. Ambas están bien documentadas con información detallada de la ubicación de cada una de las vulnerabilidades que tienen. De esta forma se pueden comparar los resultados de las herramientas y los informes del IDS.

- **DVWA** es una aplicación que está desarrollada por la compañía RandomStorm y es de código abierto. Está desarrollada en PHP y como gestor de base de datos utiliza MySQL. Se utiliza para formar desarrolladores y usuarios que deseen mejorar sus habilidades, o aprender sobre la forma en la que actúa cada una de las vulnerabilidades que contiene la aplicación [Ran16]. En la Tabla 10.2 se detallan las vulnerabilidades con las que cuenta DVWA.

Tabla 10.2: Vulnerabilidades presentes en DVWA

Vulnerabilidad	**Cantidad**
XSS reflejado (RXSS)	1
XSS persistente (PXSS)	1
SQLI	2
BSQLI	1
CSRF	1
Inclusión local de archivos (*Local File Inclusion* (LFI))	1
Inyección de comandos del sistema (*Executing Command Prompt* (CMDExec))	1

- **WackoPicko** es una aplicación vulnerable desarrollada por Adam Doupé y utilizada en [DCV10]. Su diseño fue pensado para simular una aplicación que contuviera características de una aplicación real. Actúa como una aplicación de compartición y venta de imágenes. Contiene una página de autenticación para el inicio de sesión de los usuarios y permite comentar cada imagen disponible y subir fotografías. También cuenta con un área de administración que sólo permite la creación de nuevos usuarios al sitio [Dou16]. Las vulnerabilidades incluidas en esta aplicación están muy bien documentadas y se describen en la Tabla 10.3.

Tabla 10.3: Vulnerabilidades presentes en WackoPicko

Vulnerabilidad	**Cantidad**
RXSS	3
PXSS	2
SQLI almacenado (*Stored SQL Injection* (SSQLI))	1
SQLI reflejado (*Reflected SQL Injection* (RSQLI))	1
Escalado de directorios (*Path Traversal*)	1
CMDExec	1
Inclusión de archivos (*File Inclusion* (FI))	1
Manipulación de parámetros (*Parameter Manipulation*)	1
Fallo de lógica (*Logic Flaw*)	1
Credenciales débiles (*Weak username/password*)	1
Enumeración de recursos (*Forceful Browsing*)	1
Id de sesión predecible (*SessionID vulnerability*)	1

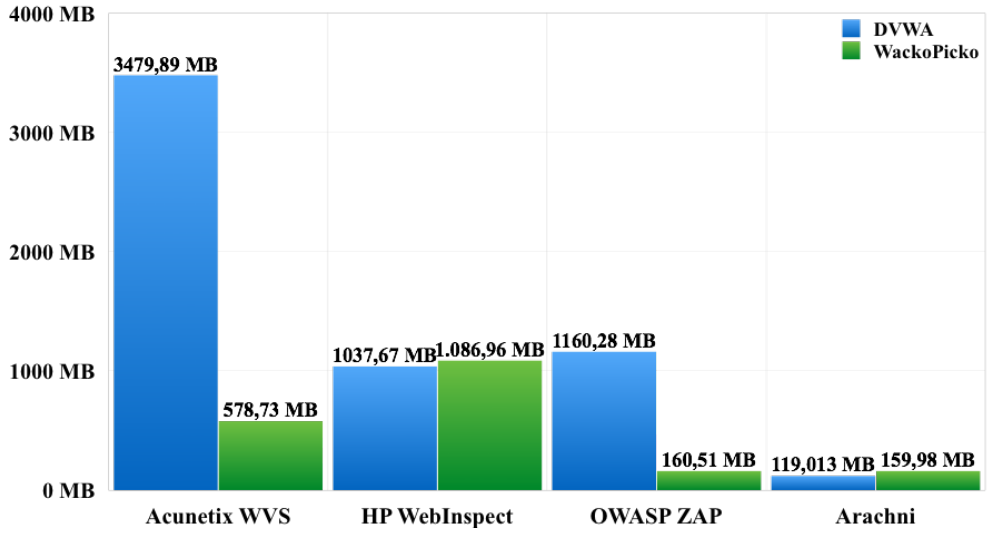
10.4. Análisis de Resultados

En cada una de las pruebas realizadas se comparan los informes generados por las alertas de SNORT, las vulnerabilidades que cada aplicación contiene y el informe de cada herramienta.

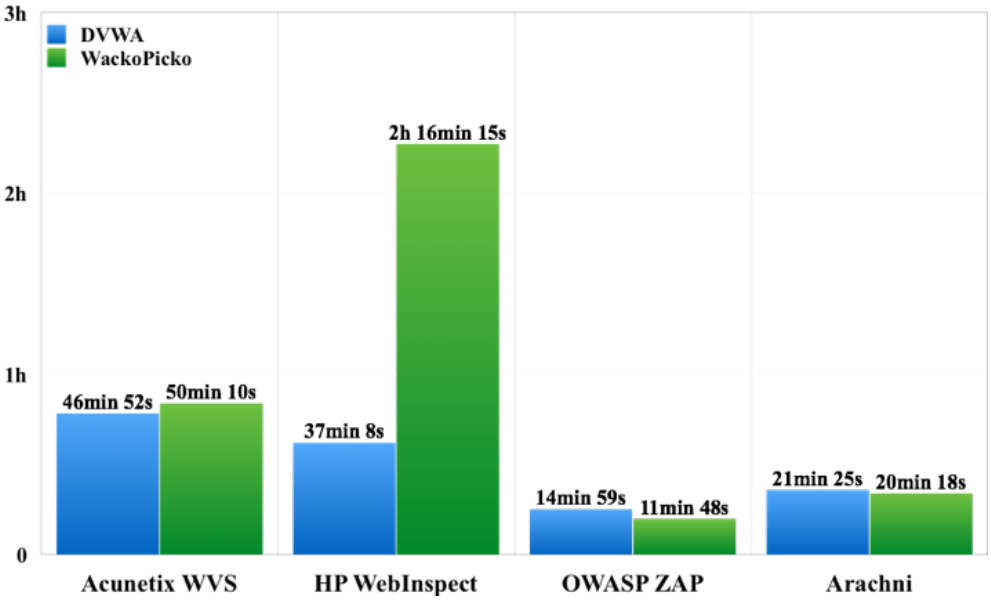
Las reglas que se utilizaron en SNORT no permitían el reconocimiento de ataques específicos de tipo persistente o reflejado para el caso de las vulnerabilidades XSS y SQLI, ya que no se ha contado con reglas que determinen el tipo de estos ataques. SNORT generó alertas que reconocían estos ataques de forma general, a excepción del tipo BSQLI, para el cual sí se cuenta con reglas adecuadas que lo identifican de forma específica y no de forma general. En cada caso, se han unificado los resultados que SNORT ha reportado como una alerta de ataque de tipo XSS o de tipo SQLI, con la debida excepción de BSQLI.

10.4.1. Tiempo y Uso de Recursos

El tráfico generado por cada una de las herramientas en este trabajo fue recogido y comparado como se muestra en la Figura 10.1(a). En la misma se puede apreciar la diferente cantidad de tráfico que genera cada herramienta, destacando principalmente el generado por Acunetix WVS en el análisis de la aplicación DVWA.



(a) Uso de recursos de red en el análisis de cada herramienta



(b) Tiempo empleado en el análisis de cada herramienta

Figura 10.1: Recursos utilizados en el análisis por cada herramienta
Recursos

Durante la ejecución de cada herramienta el tráfico generado no fue motivo de sobrecarga o lentitud para la herramienta o de no disponibilidad del sitio atacado. Por lo tanto, no es un factor determinante, en este caso particular, para el rendimiento de la

herramienta.

El tiempo empleado por cada una de las herramientas en cada análisis está muy influenciado por el tiempo que la herramienta emplea en realizar el reconocimiento del sitio, fase de rastreo, como fue el caso de la herramienta HP Webinspect, que ha obtenido un tiempo de ejecución considerablemente elevado, como muestra la Figura 10.1(b).

El tiempo de ejecución de una herramienta es un factor importante y la configuración previa al análisis incide mucho sobre el tiempo que pueda tardar en completar las fases de análisis. Cada una de las herramientas utilizadas en estas pruebas se puede configurar para acotar la profundidad de búsqueda en la aplicación y además permite definir previamente valores de entrada para los campos de los formularios que tenga la aplicación.

10.4.2. Sesiones

Las peticiones que cada aplicación lanza en los análisis se ha calculado sumando el total de alertas que Snort detectaba. Esto permite conocer la cantidad de veces en las que se ha realizado exclusivamente intentos de ataque.

Como se muestra en la Figura 10.2, la cantidad de peticiones generada no es proporcional al tiempo o al flujo de red generado en ninguno de los casos. Un claro ejemplo es el presentado por Arachni, que a pesar de su corto tiempo de ejecución, presenta el mayor número de peticiones de todas las herramientas en ambas aplicaciones.

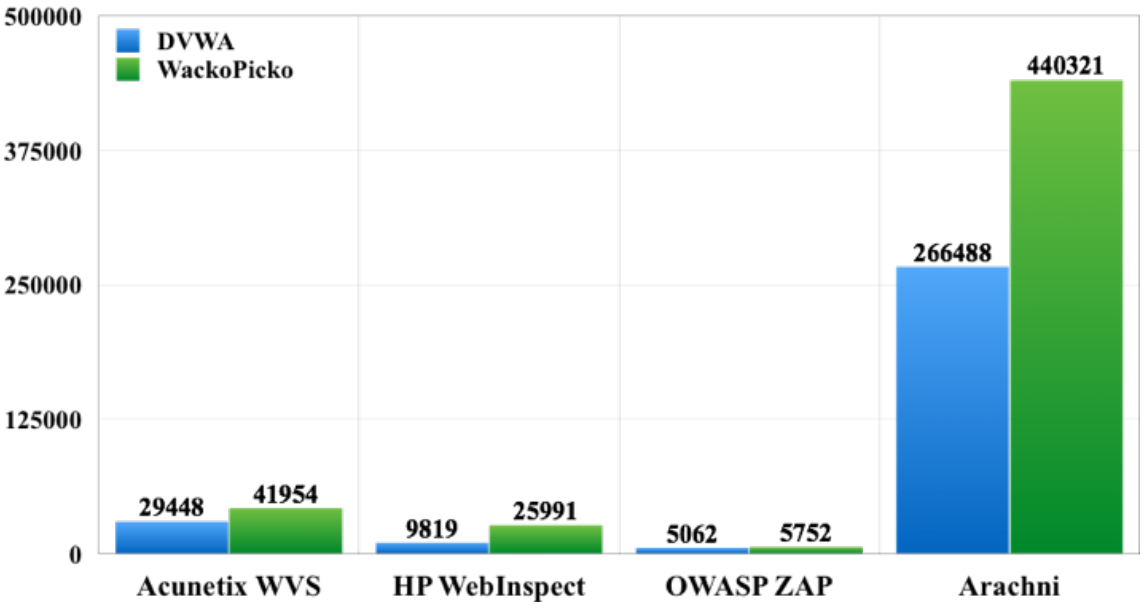


Figura 10.2: Sesiones generadas por cada herramienta en el análisis

10.4.3. Resultados DVWA

Los resultados del análisis de DVWA con todas las herramientas de este trabajo se muestra en la Tabla 10.4. En la primera columna de esta Tabla se indican las vulnerabilidades existentes en la aplicación. A continuación se muestra, para cada herramienta, si ha sido detectada por el IDS y el número de detecciones por parte de la herramienta. Si una herramienta debe de ser capaz de detectar una vulnerabilidad, al incorporar la capacidad de detectarla, está señalado en azul. Como puede verse existen ataques confirmados por el IDS que las herramientas no reportan, además de vulnerabilidades no probadas por las herramientas, a pesar de tener la capacidad de hacerlo.

En el análisis de la aplicación DVWA con Acunetix WVS se han obtenido resultados en donde resalta que no se informe de la vulnerabilidad LFI, la cual sí está presente en DVWA y según las alertas de SNORT se ha explotado durante el análisis. A pesar de ello, Acunetix WVS no lo ha incluido en su informe.

Los resultados con la herramienta OWASP ZAP no distan de los obtenidos en trabajos anteriores [MK15] pero del informe de Snort se observan detecciones de algunas vulnerabilidades que no se encuentran presentes en la aplicación.

A pesar de ser el análisis de DVWA con HP Webinspect el que más ha tardado en finalizar, ha obtenido muy pocos aciertos en comparación a los obtenidos por el resto de herramientas. Al igual que con OWASP ZAP, la herramienta no ha realizado un análisis sobre al menos todas las vulnerabilidades presentes en el OWASP Top 10 2013 [OWA13]. Por otro lado, la cantidad de falsos positivos en su informe es la menor de todas.

En el análisis realizado con Arachni sobre DVWA, la herramienta ha identificado cinco de las vulnerabilidades totales, con un bajo número de falsos positivos. SNORT en este caso ha identificado ataques de los tipos SQLI y CMDExec, pero Arachni no los muestra en su informe.

10.4.4. Resultados WackoPicko

Los resultados obtenidos del análisis de Wackopicko se pueden observar en la Tabla 10.5. En la primera columna de esta Tabla se indican las vulnerabilidades existentes en la aplicación. A continuación se muestra, para cada herramienta, si ha sido detectada por el IDS y el número de detecciones por parte de la herramienta. Si una herramienta debe de ser capaz de detectar una vulnerabilidad, al incorporar la capacidad de detectarla, está señalado en azul. Como puede verse existen ataques confirmados por el IDS que las herramientas no reportan, además de vulnerabilidades no probadas por las herramientas, a pesar de tener la capacidad de hacerlo.

La precisión en identificar las vulnerabilidades de esta aplicación con Acunetix WVS resultó ser menor al 50 % del total de vulnerabilidades que presenta WackoPicko. Además, no informa de la presencia de dos vulnerabilidades críticas, Escalado de Directorios y CMDExec, que según las alertas de SNORT fueron atacadas. La herramienta no puede determinar la categoría de los ataques del tipo SQLI y es por eso por lo que en la tabla comparativa se unifican las filas, igual a lo ocurrido con el informe de SNORT.

El informe resultante del análisis con OWASP ZAP incluye menos aciertos de detección de vulnerabilidades quee Acunetix WVS y no incluye la vulnerabilidad Escalado de Directorios, a pesar de ser explotada de acuerdo a la información que da SNORT.

HP Webinspect no presenta mayor precisión que el resto de herramientas. En su informe no separa el tipo de vulnerabilidad hallada para las vulnerabilidades SQLI. A pesar de ello el número de aciertos es mucho menor que en el resto de las herramientas. En su informe SNORT indica que hubo ataques de tipo FI y CMDExec, pero HP Webinspect no reportó la presencia de este tipo de vulnerabilidades, a pesar de que WackoPicko sí las incluye. Al igual que el resto de herramientas HP Webinspect no realiza una búsqueda de al menos todas las vulnerabilidades presentes en el Top 10 de OWASP 2013 [OWA13].

En el informe presentado por Arachni se detectaron cinco vulnerabilidades en WackoPicko, todas ellas atacadas según el informe de SNORT, no habiendo alertas en el IDS que no fueran reportadas por la herramienta. Lo que sí reporta Arachni es un alto número de falsos positivos.

Tabla 10.4: Resultados de analizar DVWA con todas las herramientas

Vulnerabilidad	Aplicación	IDS	Acunetix WVS	IDS	OWASP ZAP	IDS	HP Webinspect	IDS	Arachni
RXSS	1	X	1	X	2	X	5	X	1
PXSS	1		4		1		-		2
SQLI	2	X	4	X	1	X	3	X	-
BSQLI	1	X	3	-	-	X	-	-	-
CSRF	1	-	-	-	-	-	-	-	-
LFI	1	X	-	-	-	-	-	-	1
CMDExec	1	X	2	X	1	X	1	X	-
Escalado de directorios	-	X	1	X	1	X	11	X	1
Denegación de servicio (*Denial of Service* (DoS))	-	X	1	X	-	X	-	-	-
Inclusión remota de archivos (*Remote File Inclusion* (RFI))	-	-	-	X	1	-	-	X	1

Tabla 10.5: Resultados de analizar WackoPicko con todas las herramientas

Vulnerabilidad	Aplicación	IDS	Acunetix WVS	IDS	OWASP ZAP	IDS	HP Webinspect	IDS	Arachni
RXSS	3	X	5	X	6	X	5	X	7
PXSS	2		1		2		-		6
SSQLI	1	X	1	X	2	X	1	X	1
RSQLI	1								
Escalado de directorios	1	X	-	X	-	X	1	-	-
CMDExec	1	X	-	-	-	X	-	X	1
FI	1	-	2	X	1	X	-	-	1
Manipulación de parámetros	1	-	-	-	-	-	-	-	-
Fallo de lógica	1	-	-	-	-	-	-	-	-
Credenciales débiles	1	-	1	-	-	-	-	-	-
Enumeración de recursos	1	-	-	-	-	-	-	-	-
Id de sesión predecible	1	-	-	-	-	-	-	-	-

10.5. Síntesis del Capítulo

El objetivo de este capítulo ha sido presentar el resultado de la evaluación de las capacidades de las herramientas de análisis. El enfoque propuesto en este trabajo ha permitido obtener detalles sobre los resultados que en trabajos anteriores no se consideraron. Esto es posible gracias al uso de un sistema IDS entre la herramienta de análisis y la aplicación vulnerable que da información sobre lo que realmente está haciendo dicha herramienta.

En los resultados se muestra que las herramientas realizan ataques, confirmados por SNORT, pero que en el informe final las herramientas no lo reportan como vulnerabilidades, a pesar de su existencia en la aplicación. También, por otro lado, se prueba que a pesar de que las herramientas incorporan capacidades de detección de ciertas vulnerabilidades, no realizan ataques sobre las aplicaciones que sí son vulnerables a ellas. De esta forma se obtienen dos puntos de mejora de las herramientas:

1. Mejorar el análisis de las respuestas para detectar vulnerabilidades probadas y existentes.

2. Mejorar el criterio que determina qué pruebas realizar en cada situación.

Capítulo 11

Conclusiones y Trabajo Futuro

En este trabajo se han desarrollado técnicas para mejorar las diferentes fases del análisis de vulnerabilidades en aplicaciones web con herramientas automáticas.

Se ha comenzado con una presentación de conceptos generales, como vulnerabilidad, activo o riesgo, relacionados con la seguridad de la información, incluyendo información detallada acerca de la seguridad en las aplicaciones software.

Así, se han analizado las tareas de seguridad que hay que realizar en cada etapa del ciclo de vida de desarrollo del software, destacando entre ellas las siguientes: recopilación de los requisitos legales y normativos en la fase de análisis para determinar, por ejemplo, el tipo de control de acceso o los mecanismos de cifrado a utilizar; riesgos de seguridad en la fase de diseño para definir las salvaguardas a implementar; definición de las pruebas de seguridad en la fase de construcción y prueba; y revisión de las pruebas de seguridad en la fase de implementación.

A continuación, se ha descrito el funcionamiento de las aplicaciones web, indicando su estructura habitual en tres capas (presentación, negocio y datos) y su funcionamiento a través de peticiones y respuestas HTML.

Posteriormente, se ha dado una visión general de las vulnerabilidades en las aplicaciones web y de las principales clasificaciones que las agrupan, destacando las de las organizaciones OWASP y WASC, especialmente la clasificación OWASPT10 y las vulnerabilidades que la conforman. También se han indicado las relaciones que enlazan las diferentes clasificaciones, realizadas por esas mismas organizaciones o por otros organismos o expertos.

Después se han presentado las aplicaciones vulnerables existentes, pudiendo ser aplicaciones ya desarrolladas para realizar alguna tarea o desarrolladas a propósito con vulnerabilidades, describiéndose más detalladamente este último caso.

Luego se ha visto que, dada la amplia difusión de las aplicaciones web y de su disponibilidad casi continua a un gran número de usuarios, es necesario el estudio y la mejora del análisis de vulnerabilidades en aplicaciones web. Existen principalmente dos tipos de análisis: estático, a partir del código fuente, y dinámico, accediendo a la aplicación en ejecución. Ambos se pueden realizar de forma manual, automática o combinada. En el caso del análisis dinámico, éste se realiza en dos fases: pasiva, para encontrar todo el contenido de la aplicación, y activa, para localizar las vulnerabilidades. Seguidamente,

se han descrito las características de cuatro de las principales herramientas automáticas utilizadas en el análisis dinámico.

Tras revisar los principales trabajos relacionados sobre las distintas técnicas de evaluación de herramientas de análisis de aplicaciones web, se ha realizado una agrupación de los mismos en cuatro tipos, según se utilice un criterio de evaluación predefinido o un criterio propio, y dependiendo de si se analiza una aplicación web real o únicamente se revisan las características que incluyen las herramientas de análisis. Aunque hay algún criterio previo de evaluación como WASSEC, en la mayor parte de los trabajos previos se evalúa un grupo de herramientas en la detección de un conjunto reducido de vulnerabilidades implementadas sobre una aplicación web.

Del estudio de la literatura se han obtenidos las debilidades y carencias de estas herramientas que deberían de solventarse para mejorar sus capacidades de detección. Las principales debilidades de las herramientas según estos trabajos se encuentran en la necesidad de avanzar en los flujos de las aplicaciones ejecutando el código del cliente. Con respecto a las carencias, se comprueba que no existe una lista predefinida de vulnerabilidades que deberían detectar las herramientas, es decir, de características que deberían incluir.

A continuación, se han detallado las contribuciones de esta Tesis. Así, en primer lugar, se ha descrito el método desarrollado e implementado para mejorar la fase pasiva del análisis. Este método permite la obtención automática de valores de los campos de los formularios que la aplicación acepta como válidos, ejecutando cuando es necesario el código JavaScript. Los valores de los campos de selección, como en las soluciones anteriores, se obtienen del propio formulario, pero los valores de los campos de texto se obtienen de fuentes externas de información, como son las Tablas de Fusión de Google, a diferencia de los métodos habituales que exigen su inclusión manual por parte del usuario de la herramienta de análisis. El método habitual es más eficaz ya que los valores son realmente válidos, pero es menos eficiente cuantos más formularios y campos tenga la aplicación, ya que exige la intervención del usuario y un mayor consumo de tiempo. Para mejorar la localización de valores válidos en las Tablas de Fusión se intentan buscar valores para los campos de texto a la vez, para que los resultados estén realmente relacionados, por ejemplo, una localidad con su provincia. Como en muchos casos la recuperación por parte de las aplicaciones web de los valores de los campos de selección se realiza mediante la ejecución de código JavaScript en el método desarrollado y los formularios se van rellenando y ejecutando automáticamente con un navegador de uso habitual, hasta que se consigue encontrar un conjunto de valores válidos. En las pruebas realizadas con la herramienta desarrollada se comprueba que este método permite localizar un número mayor de páginas de las aplicaciones, entre un 13 % y un 16 % más que los métodos habituales, sin que sea necesaria una revisión previa de los formularios y de los campos de la aplicación por parte del usuario que realiza el análisis.

En segundo lugar se ha elaborado un algoritmo para la obtención de una clasificación de vulnerabilidades web que unifica las clasificaciones actuales y que mantiene actualizadas las relaciones entre sus vulnerabilidades. Este algoritmo selecciona las clasificaciones más relevantes y a continuación busca las relaciones entre ellas en el siguiente orden:

relaciones proporcionadas por los propios organismos que mantienen las clasificaciones, relaciones proporcionadas por otros organismos y relaciones proporcionadas por expertos en seguridad. En el último caso las relaciones se obtienen realizando búsquedas en Internet de las palabras clave de cada vulnerabilidad junto con la codificación empleada en cada clasificación. Las palabras clave de cada vulnerabilidad son las palabras (etiquetas) que la definen en exclusiva, esto es, el conjunto de palabras que está asociado a esa vulnerabilidad y sólo a ella. La obtención de las palabras clave de cada vulnerabilidad se realiza eliminando las palabras más comunes que aparecen en las descripciones de las vulnerabilidades, quedando las más significativas. Este algoritmo se ha implementado en una página web que permite la consulta de una vulnerabilidad de alguna de las clasificaciones OWASPTGv3, OWASPTGv4, WASCTC y CWE, o de su descripción, obteniendo la correspondiente en las demás clasificaciones. El método se ha probado, obteniéndose relaciones que son reales en el 85 % de los casos. De esta forma se obtiene una clasificación de las vulnerabilidades que deben ser capaces de detectar las herramientas. Además, la información que se obtiene puede permitir a programadores y profesionales de seguridad poder relacionar las vulnerabilidades encontradas por las herramientas que utilicen con las que se encuentran en las clasificaciones.

En tercer lugar se han seleccionado, de las aplicaciones vulnerables desarrolladas a propósito, las que contiene un número mayor de vulnerabilidades. En estas aplicaciones se han añadido ocho vulnerabilidades adicionales que no tenían. El objetivo es proporcionar a estudiantes y profesionales de seguridad un marco de pruebas con el que adquirir habilidades y con el que también poder probar herramientas de análisis. Se han probado varias herramientas en las aplicaciones vulnerables, llegando a la conclusión de que estas herramientas suelen detectar únicamente las vulnerabilidades más conocidas. De las ocho vulnerabilidades añadidas, solamente una herramienta ha conseguido detectar una de ellas.

En cuarto y último lugar se ha obtenido información sobre cómo las herramientas de análisis dinámico interactúan con las aplicaciones analizadas para obtener un panorama más claro respecto al funcionamiento y precisión de las mismas. Para ello se ha aplicado un enfoque distinto al tradicional, introduciendo en la estructura típica de estudio, formada por un servidor web y un conjunto de herramientas, un sistema de detección de intrusos entre estos dos elementos. El sistema detector de intrusos se ha configurado con un conjunto de unas 7300 reglas para el tráfico exclusivamente HTTP. Una vez analizadas las aplicaciones web con las herramientas se tienen tres fuentes de información: las vulnerabilidades en las aplicaciones vulnerables, los informes de las herramientas y los informes del IDS. Comparando las tres fuentes se sabe, por un lado, si la herramienta prueba todas las vulnerabilidades de la aplicación para las que está capacitada, y por otro lado, si informa de todas las vulnerabilidades de la aplicación que efectivamente ha probado. Las pruebas realizadas con varias herramientas muestran que no prueban ni informan de todo lo que deberían, es decir, están capacitadas para una mayor tasa de detección de la que finalmente consiguen.

11.1. Trabajo Futuro

Como trabajo futuro pueden señalarse las siguientes líneas de investigación:

- **Implementar una base de conocimiento de valores y campos de los formularios**: Para poder realizar los análisis de manera más rápida, sería interesante definir e implementar una base de conocimiento de valores y campos que se podría ir completando conforme la herramienta vaya analizando distintos formularios, de manera que pueda usarse la información recopilada en análisis anteriores en los siguientes.

- **Añadir más clasificaciones de vulnerabilidades**: En la clasificación unificada de vulnerabilidades se han tenido en cuenta las cuatro clasificaciones actuales más conocidas. En el futuro deberían de añadirse nuevas clasificaciones para obtener relaciones entre todas las clasificaciones actuales.

- **Completar la aplicación vulnerable**: obtener, y mantener posteriormente, un conjunto de aplicaciones web vulnerables a propósito que contenga todas las vulnerabilidades de las clasificaciones actuales, partiendo de la clasificación unificada que se ha desarrollado y de las aplicaciones vulnerables ampliadas.

- **Formación en desarrollo seguro y pruebas de vulnerabilidades**: Utilizar la clasificación unificada de vulnerabilidades para proporcionar información a estudiantes sobre las vulnerabilidades a evitar en el desarrollo de aplicaciones o en sus métodos de detección. En este último caso se podría usar las aplicaciones vulnerables ampliadas en este trabajo. De esta forma se tendrían en cuenta eventualmente todas las vulnerabilidades y no sólo las más comunes.

- **Relacionar las vulnerabilidades de las herramientas**: Al igual que se ha obtenido una relación entre las vulnerabilidades presentes en las principales clasificaciones, también debe de definirse un método para relacionar las vulnerabilidades que detecta cada una de las herramientas automáticas, de forma que se sea posible conocer, para una vulnerabilidad dada, cómo se llama en cada herramienta en caso de que incluya la capacidad de detección.

- **Mejorar el análisis de las herramientas**: En las pruebas realizadas sobre las capacidades de las herramientas se ha usado como elemento intermedio un sistema IDS. En el futuro sería interesante incluir también un cortafuegos (*firewall*) de aplicación (*Web Application Firewall* (WAF)) y añadir nuevas reglas a estos dos elementos para obtener un análisis más detallado del tráfico y las acciones que realizan las herramientas de análisis cuando atacan las aplicaciones vulnerables. Esta mejora incluiría utilizar más aplicaciones vulnerables.

Bibliografía

[Acu15] Acunetix. Web Vulnerability Scanner v10. Product Manual, June 2015.

[Acu16] Acunetix. Annual Web App Security Report 2016. `http://www.acunetix.com/blog/news/web-app-security-report-2016/`, June 2016.

[AIM14] H. Alnabulsi, Md. R. Islam, and Q. Mamun. Detecting SQL Injection Attacks Using SNORT IDS. In *Proceedings of the 2014 Asia-Pacific World Congress on Computer Science and Engineering*, pages 1–7, Nadi, Fiji, November 2014.

[AKdLM10] R. E. Assad, T. Katter, and S. R. de Lemos Meira. Security Quality Assurance on Web-based Application Through Security Requirements Tests Based on OWASP Test Document: Elaboration, Execution and Automation. In *Proceedings of the 2nd OWASP Ibero-American Web Applications Security Conference*, Lisboa, Portugal, November 2010.

[ALY+15] F. Akowuah, J. Lake, X. Yuan, E. Nuakoh, and H. Yu. Testing the Security Vulnerabilities of OpenEMR 4.1.1: A Case Study. *Journal of Computing Sciences in Colleges*, 30(3):26–35, January 2015.

[Ara16] Arachni. ARACHNI Web Application Security Scanner Framework. `http://www.arachni-scanner.com`, March 2016.

[ASW10] A. Austin, B. Smith, and L. Williams. Towards Improved Security Criteria for Certification of Electronic Health Record Systems. In *Proceedings of the 2010 ICSE Workshop on Software Engineering in Health Care*, pages 68–73, New York, USA, May 2010.

[AW11] A. Austin and L. Williams. One Technique is Not Enough: A Comparison of Vulnerability Discovery Techniques. In *Proceedings of the 2011 International Symposium on Empirical Software Engineering and Measurement*, pages 97–106, Washington DC, USA, September 2011.

[Bar11] P. Baral. Web Application Scanners: A Review of Related Articles. *IEEE Potentials*, 30(2):10–14, March 2011.

[BBGM10] J. Bau, E. Bursztein, D. Gupta, and J. Mitchell. State of the Art: Automated Black-Box Web Application Vulnerability Testing. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, pages 332–345, Washington DC, USA, 2010.

[BFOG08] P. E. Black, E. N. Fong, V. Okun, and R. Gaucher. Software Assurance Tools: Web Application Security Scanner Functional Specification Version 1.0. NIST Special Publication 500-26, 2008.

[BK05] P. E. Black and M. Kass. Software Security Assurance Tools, Techniques and Metrics. In *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, pages 461–461, New York, USA, November 2005.

[Che12] S. Chen. General Features Comparison - Web Application Scanners. `http://www.sectoolmarket.com`, 2012.

[CHM11] M. J. Cafarella, A. Halevy, and J. Madhavan. Structured Data on the Web. *Communications of the ACM*, 54(2):72–79, February 2011.

[Con09] Web Application Security Consortium. Web Application Security Scanner Evaluation Criteria WASSEC. `http://goo.gl/aePtyC`, April 2009.

[Con10] Web Application Security Consortium. The WASC Threat Classification. `http://projects.webappsec.org/w/page/13246978/ThreatClassification`, 2010.

[Con12] Web Application Security Consortium. Threat Classification Taxonomy Cross Reference View. `http://projects.webappsec.org/w/page/13246975/Threat%20Classification%20Taxonomy%20Cross%20Reference%20View`, 2012.

[Cor15a] The MITRE Corporation. Common Attack Patterns Enumeration and Classfication. `http://capec.mitre.org/`, 2015.

[Cor15b] The MITRE Corporation. Common Weakness Enumeration. `http://cwe.mitre.org`, 2015.

[DAA13] M. Dabbour, I. Alsmadi, and E. Alsukhni. Efficient Assessment and Evaluation for Websites Vulnerabilities using SNORT. *International Journal of Security and its Applications*, 7(1):7–15, January 2013.

[DBH14] N. I. Daud, K. A. A. Bakar, and M. S. Md. Hasan. A Case Study on Web Application Vulnerability Scanning Tools. In *Proceedings of the Conference of Science and Information*, pages 595–600, London, United Kingdom, August 2014.

[DCV10] A. Doupé, M. Cova, and G. Vigna. Why Johnny Can't Pentest: An Analysis of Black-Box Web Vulnerability Scanners. In *Proceedings of the 7th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 111–131, Bonn, Germany, July 2010.

[DGdLO05] Y. Demchenko, L. Gommans, C. de Laat, and B. Oudenaarde. Web Services and Grid Security Vulnerabilities and Threats Analysis and Model. In *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, pages 262–267, Washington DC, USA, November 2005.

[Dou16] A. Doupé. WackoPicko Vulnerable Website. `https://github.com/adamdoupe/WackoPicko`, March 2016.

[Fir16] I. Firns. Dedicated spooler for Snort's Unified2 Binary Output Format. `https://github.com/firnsy/`, April 2016.

[FK11] A. M. Ferreira and H. Klepee. Effectiveness of Automated Application Penetration Testing Tools. Technical Report, University of Amsterdam, April 2011.

[FO07] E. Fong and V. Okun. Web Application Scanners: Definitions and Functions. In *Proceedings of the 40th Annual Hawaii International Conference on System Sciences*, pages 280b–280b, Washington DC, USA, January 2007.

[FVM07] J. Fonseca, M. Vieira, and H. Madeira. Testing and Comparing Web Vulnerability Scanning Tools for SQL Injection and XSS Attacks. In *13th Pacific Rim International Symposium on Dependable Computing*, pages 365–372, Melbourne, Australia, December 2007.

[FVM14] J. Fonseca, M. Vieira, and H. Madeira. Evaluation of Web Security Mechanisms Using Vulnerability amp; Attack Injection. *IEEE Transactions on Dependable and Secure Computing*, 11(5):440–453, September 2014.

[GHJ+10] H. Gonzalez, A. Y. Halevy, C. S. Jensen, A. Langen, J. Madhavan, R. Shapley, W. Shen, and J. Goldberg-Kidon. Google Fusion Tables: Web-centered Data Management and Collaboration. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, pages 1061–1066, New York, USA, June 2010.

[Goo16] Google Inc. Google Gruyere. `https://google-gruyere.appspot.com/`, March 2016.

[GS11] S. Gupta and L. Sharma. Analysis and Assessment of Web Application Security Testing Tools. In *Proceedings of the 5th National Conference on Computing for Nation Develpoment*, pages 1–2, New Delhi, India, March 2011.

[HHLT03] Y. Huang, S. Huang, T. Lin, and C. Tsai. Web Application Security Assessment by Fault Injection and Behavior Monitoring. In *Proceedings of the 12th International Conference on World Wide Web*, pages 148–159, New York, USA, May 2003.

[HP15] HP. HP WebInsPect. Product Manual, HP, March 2015.

[HPH+16] C. Henard, M. Papadakis, M. Harman, Y. Jia, and Y. Le Traon. Comparing White-box and Black-box Test Prioritization. In *Proceedings of the 38th International Conference on Software Engineering*, pages 523–534, New York, USA, May 2016.

[Int05] International Organization for Standardization. International Standard ISO/IEC 27001, 2005.

[Int17] InternetLiveStats.com. Internet Live Stats. `http://www.internetlivestats.com/internet-users/`, 2017.

[JLM14] Y. Jiang, X. Li, and W. Meng. DiscWord: Learning Discriminative Topics. In *Proceedings of the 2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technologies*, volume 2, pages 63–70, Warsaw, Poland, August 2014.

[Kha10] M. E. Khan. Different Forms of Software Testing Techniques for Finding Errors. *International Journal of Computer Science Issues*, 7(3):11–16, 2010.

[KLD10] P. K. Keong Loh and Subramanian D. Fuzzy Classification Metrics for Scanner Assessment and Vulnerability Reporting. *IEEE Transactions on Information Forensics and Security*, 5(4):613–624, December 2010.

[KSG+14] A. Khalili, A. Sami, M. Ghiasi, S. Moshtari, Z. Salehi, and M. Azimi. Software Engineering Issues Regarding Securing ICS: An Industrial Case Study. In *Proceedings of the 1st International Workshop on Modern Software Engineering Methods for Industrial Automation*, pages 1–6, New York, USA, May 2014.

[KZLR11] N. Khoury, P. Zavarsky, D. Lindskog, and R. Ruhl. An Analysis of Black-Box Web Application Security Scanners against Stored SQL Injection. In *Proceedings of the IEEE Third International Conference on Privacy, Security, Risk and Trust and IEEE Third Inernational Conference on Social Computing*, pages 1095–1101, Boston, USA, October 2011.

[Lab12] Laboratory Synonyms. Synonymlab. `http://www.synonymlab.com`, 2012.

[LL07] B. Liu and Y. Li. A Normalized Levenshtein Distance Metric. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 29:1091–1095, 2007.

[Mar12] J. Martirosyan. Security Evaluation of Web Application Vulnerability Scanners' Strengths and Limitations Using Custom Web Application. Master Thesis, California State University - East Bay, October 2012.

[MB08] R. A. Martin and S. Barnum. Common Weakness Enumeration (CWE) Status Update. *ACM SIGAda Ada Letters*, XXVIII(1):88–91, April 2008.

[Mcq14] K. Mcquade. Open Source Web Vulnerability Scanners: The Cost Effective Choice? In *Proceedings of the Conference for Information Systems Applied Research*, pages 1–13, Baltimore, USA, October 2014.

[MK15] Y. Makino and V. Klyuev. Evaluation of Web Vulnerability Scanners. In *Proceedings of the IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications*, volume 1, pages 399–402, Warsaw, Poland, September 2015.

[MKK+08] J. Madhavan, D. Ko, L. Kot, V. Ganapathy, A. Rasmussen, and A. Halevy. Google's Deep Web Crawl. *Proceedings of the Very Large Data Bases Endowment*, 1(2):1241–1252, August 2008.

[ND12] S. Nidhra and J. Dondeti. Blackbox and Whitebox Testing Techniques - A Literature Review. *International Journal of Embedded Systems and Applications*, 2(2):29–50, June 2012.

[Net17] Netcraft. January 2017 Web Server Survey. `https://news.netcraft.com/archives/category/web-server-survey/`, 2017.

[NT10] C. S. Nuno Teodoro. Automating Web Applications Security Assessments through Scanners. In *Proceedings of the OWASP Ibero-American Web Applications Security Conference*, Lisboa, Portugal, December 2010.

[Ope13] OpenCart. Open Source Shopping Cart Solution. `https://www.opencart.com/`, 2013.

[OWA08] OWASP. Open Web Application Security Project: Owasp testing guide v3. `https://www.owasp.org`, 2008.

[OWA13] The Open Web Application Security Project OWASP. OWASP Top 10 - 2013 The Ten Most Critical Web Application Security Risks. Release, The Open Web Application Security Project OWASP, June 2013.

[OWA14] OWASP. Open Web Application Security Project: Owasp testing guide v4. `https://www.owasp.org`, 2014.

[OWA16a] OWASP. OWASP Mutillidae II Project. `https://www.owasp.org/index.php/OWASP_Mutillidae_2_Project`, March 2016.

[OWA16b] OWASP. OWASP WebGoat Project. `https://www.owasp.org/index.php/Category:OWASP_WebGoat_Project`, March 2016.

[Par15] S. Parmar. Vulnerability Checker for Infosecurity. *International Journal of Science and Research*, 4(3):1593–1596, 2015.

[PZK15] M. Parvez, P. Zavarsky, and N. Khoury. Analysis of Effectiveness of Black-box Web Application Scanners in Detection of Stored SQL Injection and Stored XSS Vulnerabilities. In *Proceedings of the 10th International Conference for Internet Technology and Secured Transactions*, pages 186–191, London, United Kingdom, December 2015.

[Ran16] RandomStorm. Damn Vulnerable Web Application (DVWA). `http://www.dvwa.co.uk`, March 2016.

[RMnAVGV16] F. Román Muñoz, E. A. Armas Vega, and L. J. García Villalba. Analyzing the Traffic of Penetration Testing Tools with an IDS. *The Journal of Supercomputing*, pages 1–16, November 2016.

[RMnGV12] F. Román Muñoz and L. J. García Villalba. Preproceso de Formularios para el Análisis de Seguridad de las Aplicaciones Web. In *Actas de la XII Reunión Española sobre Criptología y Seguridad de la Información*, San Sebastián, Spain, September 2012.

[RMnGV13] F. Román Muñoz and L. J. García Villalba. Methods to Test Web Applications Scanners. In *Proceedings of the 6th International Conference on Information Technology*, Amman, Jordan, May 2013.

[RMnGV14] F. Román Muñoz and L. J. García Villalba. Capacidades de Detección de las Herramientas de Análisis de Vulnerabilidades en Aplicaciones Web. In *Actas del XIII Reunión Española sobre Criptología y Seguridad de la Información*, Alicante, Spain, September 2014.

[RMnGV15a] F. Román Muñoz and L. J. García Villalba. Algoritmo para el Mapeo de Clasificaciones de Vulnerabilidades Web. In *Actas del VIII Congreso Iberoamericano de Seguridad Informática*, Quito, Ecuador, November 2015.

[RMnGV15b] F. Román Muñoz and L. J. García Villalba. Búsqueda de Relaciones entre Vulnerabilidades de Aplicaciones Web. In *Actas del VIII Congreso Iberoamericano de Seguridad Informática*, Quito, Ecuador, September 2015.

[RMnGV15c] F. Román Muñoz and L. J. García Villalba. Web from Preprocessor for Crawling. *Multimedia Tools and Applications*, 74(19):8559–8570, October 2015.

[RMnGV16] F. Román Muñoz and L. J. García Villalba. An Algorithm to Find Relationships Between Web Vulnerabilities. *The Journal of Supercomputing*, pages 1–29, June 2016.

[RMnSCGV16] F. Román Muñoz, I. I. Sabido Cortés, and L. J. García Villalba. Aplicaciones Web Vulnerables a Propósito. In *Actas de las II Jornadas Nacionales de Investigación en Ciberseguridad*, pages 130–135, Granada, Spain, June 2016.

[RMnSCGV17] F. Román Muñoz, I. I. Sabido Cortes, and L. J. García Villalba. Enlargement of Vulnerable Web Applications for Testing. *The Journal of Supercomputing*, pages 1–20, February 2017.

[Roe16] M. Roesch. Snort - Network Intrusion Detection and Prevention System. `https://www.snort.org/`, Abril 2016.

[Sae14a] F. A. Saeed. Using WASSEC to Analysis and Evaluate Open Source Web Application Security Scanners, April 2014.

[Sae14b] F. A. Saeed. Using WASSEC to Evaluate Commercial Web Application Security Scanners. *International Journal of Soft Computing and Engineering*, 4(1):177–181, March 2014.

[SAN11] SANS. CWE/SANS TOP 25 Most Dangerous Software Errors. `http://www.sans.org/top25-software-errors`, 2011.

[Sec16] WhiteHat Security. Web Applications Security Statistics Report. Release, WhiteHat Security, 2016.

[SM15] A. Sagala and E. Manurung. Testing and Comparing Result Scanning Using Web Vulnerability Scanner. *Advanced Science Letters*, 21(11):3458–3462, November 2015.

[Sof16] Soft112. The ButterFly - Security Project. `http://the-butterfly-security-project.soft112.com/`, March 2016.

[Sub16] Subgraph. Vega Vulnerability Scanner. `https://subgraph.com/vega/`, April 2016.

[Sut10] L. Suto. *Analyzing the Accuracy and Time Costs of Web Application Security Scanners*. February 2010.

[The12] The Open Web Application Security Project OWASP. OWASP Zed Attack Proxy Project. `https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project`, April 2012.

[The13] The Open Web Application Security Project OWASP. OWASP Top 10 - 2013 The Ten Most Critical Web Application Security Risks, June 2013.

[The16] The Open Web Application Security Project OWASP. OWASP Zed Attack Proxy Project. `https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project`, April 2016.

[TWG13] O. Tripp, O. Weisman, and L. Guy. Finding Your Way in the Testing Jungle: A Learning Approach to Web Security Testing. In *Proceedings of the 2013 International Symposium on Software Testing and Analysis*, pages 347–357, New York, NY, USA, July 2013.

[Wil16] D. WillisWebber. Ruby on Rails Application for Network Security Monitoring. `https://github.com/Snorby/snorby`, April 2016.

[Wri08] A. Wright. Searching the Deep Web. *Communications of the ACM*, 51(10):14–15, October 2008.

[XTH08] P. Xiang, K. Tian, and Q. Huang. A Framework of Deep Web Crawler. In *Proceedings of the 27th Chinese Control Conference*, pages 582–586, Kunming, China, July 2008.

Parte II

Publicaciones

Apéndice A

Lista de Publicaciones

1. Fernando Román Muñoz, Luis Javier García Villalba: Preproceso de Formularios para el Análisis de Seguridad de las Aplicaciones Web. Actas de la XII Reunión Española sobre Criptología y Seguridad de la Información (RECSI 2012), pages 1–6, Donostia–San Sebastián, España, Septiembre 4 – 7, 2012.

2. Fernando Román Muñoz, Luis Javier García Villalba: Methods to Test Web Applications Scanners. Proceedings of the 6th International Conference on Information Technology (ICIT 2013), Amman, Jordan, pages 1–6, May 8 – 10, 2013.

3. Fernando Román Muñoz, Luis Javier García Villalba: Capacidades de Detección de las Herramientas de Análisis de Vulnerabilidades en Aplicaciones Web. Actas de la XIII Reunión Española sobre Criptología y Seguridad de la Información (RECSI 2014), pages 157–161, Alicante, España, Septiembre 2 – 5, 2014.

4. Fernando Román Muñoz, Luis Javier García Villalba: Web from Preprocessor for Crawling. Multimedia Tools and Applications, 74(9): 8559–8570, October 2015.

5. Fernando Román Muñoz, Luis Javier García Villalba: Algoritmo para el Mapeo de Clasificaciones de Vulnerabilidades Web. Actas del VIII Congreso Iberoamericano de Seguridad Informática (CIBSI 2015), pages 1–6, Quito, Ecuador, Noviembre 10–12 2015.

6. Fernando Román Muñoz, Luis Javier García Villalba: Búsqueda de Relaciones entre Vulnerabilidades de Aplicaciones Web. Actas del VIII Congreso Iberoamericano de Seguridad Informática (CIBSI 2015), Quito, pages 1–7, Ecuador, Noviembre 10–12 2015.

7. Fernando Román Muñoz, Iván Israel Sabido Cortes, Luis Javier García Villalba: Aplicaciones Web Vulnerables a Propósito. Actas de las II Jornadas Nacionales de Investigación en Ciberseguridad (JNIC 2016), pages 130–135, Granada, España, Junio 15 – 17, 2016.

8. Fernando Román Muñoz, Luis Javier García Villalba: An Algorithm to Find Relationships between Web Vulnerabilities. The Journal of Supercomputing: 1–29, June 2016.

9. Fernando Román Muñoz, Esteban Alejandro Armas Vega, Luis Javier García Villalba: Analyzing the Traffic of Penetration Testing Tools with an IDS. The Journal of Supercomputing: 1–16, November 2016.

10. Fernando Román Muñoz, Iván Israel Sabido Cortes, Luis Javier García Villalba: Enlargement of Vulnerable Web Applications for Testing. The Journal of Supercomputing: 1–20, February 2017.